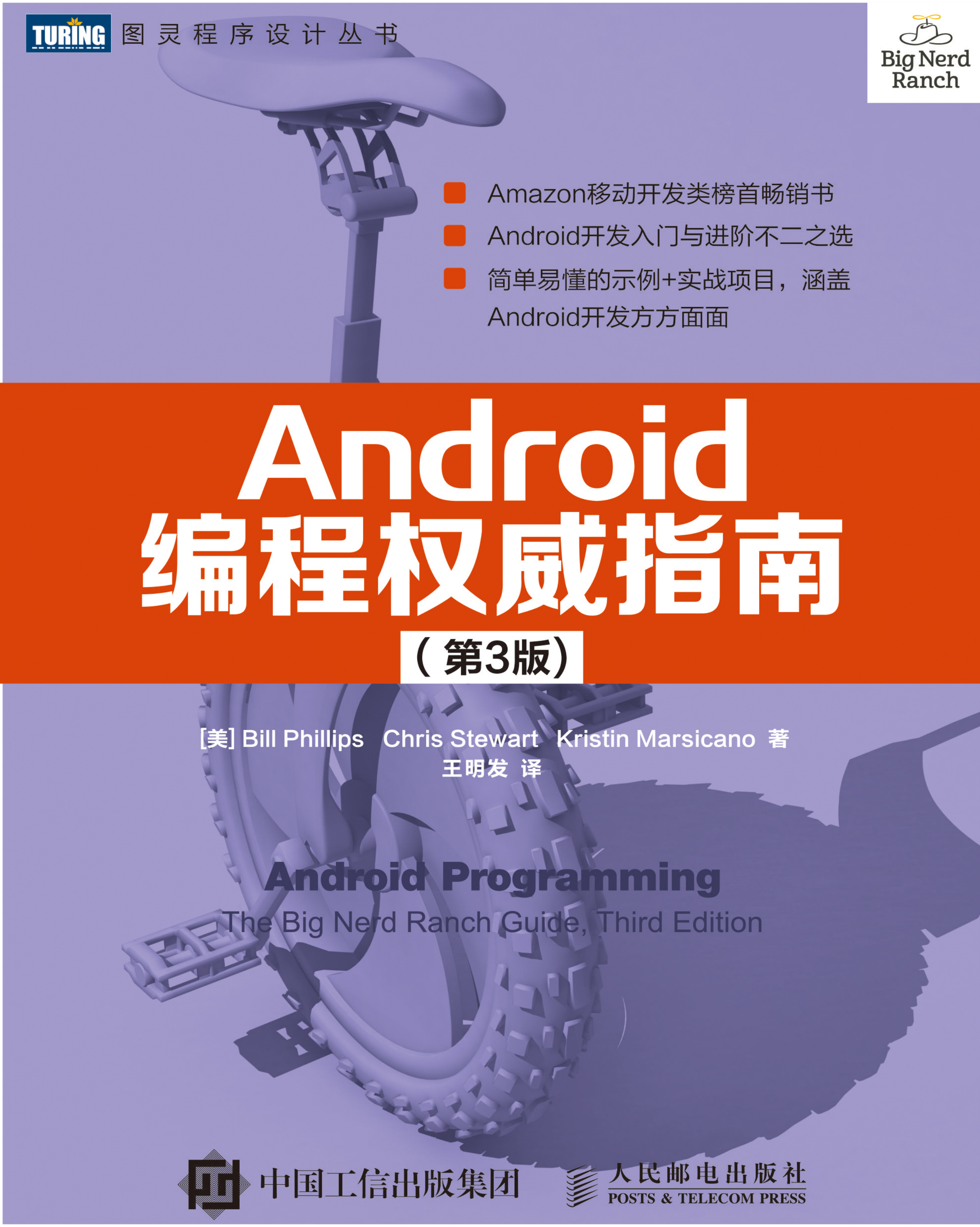
TURING 图灵程序设计丛书

Big Nerd Ranch



- Amazon移动开发类榜首畅销书
- Android开发入门与进阶不二之选
- 简单易懂的示例+实战项目，涵盖Android开发方方面面

Android编程权威指南

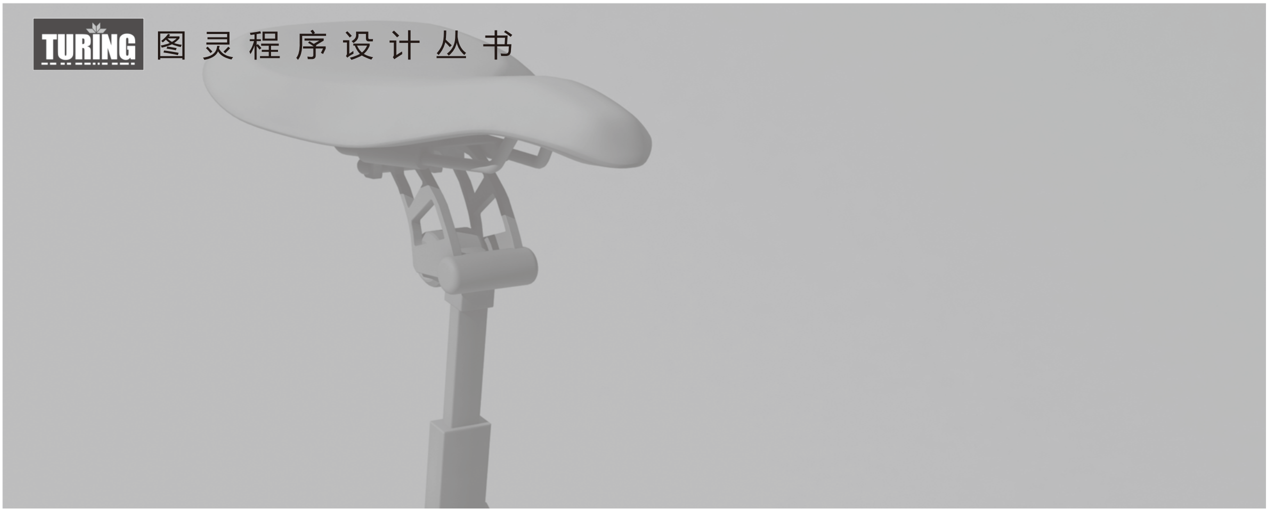
（第3版）

[美] Bill Phillips　Chris Stewart　Kristin Marsicano 著
王明发 译

Android Programming

The Big Nerd Ranch Guide, Third Edition

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS



TURING 图灵程序设计丛书

Android编程权威指南

（第3版）

[美] Bill Phillips　Chris Stewart　Kristin Marsicano 著
王明发 译



Android Programming

The Big Nerd Ranch Guide, Third Edition

人民邮电出版社
北京

图书在版编目（C I P）数据

Android编程权威指南 / （美）比尔·菲利普斯
(Bill Phillips)，（美）克里斯·斯图尔特
(Chris Stewart)，（美）克莉丝汀·马西卡诺
(Kristin Marsicano) 著 ; 王明发 译. -- 3版. -- 北
京 : 人民邮电出版社, 2017.6
（图灵程序设计丛书）
ISBN 978-7-115-45759-2

Ⅰ. ①A… Ⅱ. ①比… ②克… ③克… ④王… Ⅲ. ①
移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2017)第114263号

内 容 提 要

Big Nerd Ranch 是美国一家专业的移动开发技术培训机构。本书主要以其 Android 训练营教学课程为基础，融合了几位作者多年的心得体会，是一本完全面向实战的 Android 编程权威指南。全书共 36 章，详细介绍了 8 个 Android 应用的开发过程。通过这些精心设计的应用，读者可掌握很多重要的理论知识和开发技巧，获得宝贵的开发经验。

第 3 版较之前版本增加了对数据绑定等新工具的介绍，同时新增了针对单元测试、辅助功能和 MVVM 架构等主题的章节。如果你熟悉 Java 语言，或者了解面向对象编程，那就立刻开始 Android 编程之旅吧！

◆ 著　　[美] Bill Phillips　Chris Stewart　Kristin Marsicano
译　　王明发
责任编辑　谢婷婷
责任印制　彭志环

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　印刷

◆ 开本：800×1000　1/16
印张：37.25　彩插：1
字数：884千字　2017年 6 月第 3 版
印数：38 501－42 000册　2017年 6 月北京第 1 次印刷

著作权合同登记号　图字：01-2017-3139号

定价：129.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

广告经营许可证：京东工商广登字 20170147 号

献　　词

献给我桌上的唱片机。感谢你陪伴我完成本书。我保证，很快你就会有新唱针了。

——B.P.

献给我的爸爸David，他教我懂得辛苦工作的意义。献给我的妈妈Lisa，她一直督促我去做正确的事。

——C.S.

献给我的爸爸Dave Vadas，他激励并支持我投身计算机行业。献给我的妈妈Joan Vadas，在这么多年的浮浮沉沉里，她总能让我保持乐观。（她会给我支招：心情不好的时候就看集《黄金女郎》吧！）

——K.M.

致　　谢

这是本书第3版。我们常说，当然，也应再三强调：仅凭作者是无法成书的。这背后是团队的力量：合作者、责任编辑和支持者。没有他们，想抓住重点并撰写出这么多的出版素材肯定不可能。

- Brian Hardy和Bill很有雄心，他们从无到有，写出了本书第1版。真了不起。
- 感谢我们Android开发团队的同事Andrew Lunsford、Bolot Kerimbaev、Brian Gardner、David Greenhalgh、Josh Skeen、Matt Compton、Paul Turner和Rashad Cureton。他们一直用这些还不够完善的材料教学，并提出了不少宝贵建议，也修改了一些错误。能和这样有趣、有才的团队一起工作，此生无憾。在Big Nerd Ranch工作的日子，每一天都是享受。
- 特别感谢Andrew。他为本书同步更新了一大批Android Studio截图。他很细心，不放过任何细节，说话还幽默辛辣，令人欣赏。
- Zack Simon，说起话来轻声细语，是我们Big Nerd Ranch了不起的天才设计师。他不声不响地更新了附在书后的Android开发速查表，给了我们一个大大的惊喜。Zack，谢谢你！如果速查表用着不错，你也去谢谢他吧！
- 感谢Kar Loong Wong，他重新设计了crime应用列表屏。只要他多伸手，本书的应用肯定会越来越好看。
- 感谢Mark Dalrymple，他审阅了constraint layout这部分内容，使之更加准确、完善。凑巧碰到他的话，如果你也在搞constraint layout，千万记得请他把关，他可是这方面的专家。不搞也没关系，可以请多才多艺的他扎些气球小动物玩玩。
- 感谢Aaron Hillegass。他若不创建Big Nerd Ranch公司，这一切都无从谈起。
- 感谢我们的编辑Elizabeth Holaday。据说，著名的"垮掉的一代"文学作家William S. Burroughs有时会把自己的作品分成多个部分，抛向空中，然后以稿件的落地顺序出书。要不是Liz，在遇到困惑、一时冲动时，相信我们也会这么做。在她的指导下，我们才能有的放矢，写出清晰、简洁的书稿。
- 感谢Ellie Volckhausen为本书设计了封面。
- 感谢我们的文字编辑Anna Bentley和审稿编辑Simone Payment。感谢她俩的打磨完善。
- 感谢IntelligentEnglish.com网站的Chris Loper。他设计并制作了本书的纸质版和电子版。他的DocBook工具简直太好用了。

最后感谢我们的学员。我们之间有个反馈环：我们以本书内容教学，他们不断给予反馈。没有这个反馈环，就没有这本书，即便有，也不会越来越完善。如果说Big Nerd Ranch公司的图书够特别（希望如此），功劳就在于这个反馈环。再次感谢。

如何学习Android开发

学习Android开发，对每个新手都是一个很大的挑战，就好像在异国他乡学会生存一样。即使会说当地的语言，一开始也绝不会有在家的感觉，因为你不能完全理解周围的人所理解的东西。原有的知识储备在新环境下可能完全派不上用场。

Android有自己的语言文化——使用Java语言。但仅掌握Java远远不够，还需要通过学习很多新的理论和技术知识来理清头绪，从而指引你穿越陌生的领域。

该我们登场了。在Big Nerd Ranch，我们认为，要成为Android开发人员，必须做到：

- 着手开发一些Android应用；
- 彻底理解你的Android应用。

本书将协助你完成以上两件事情。我们已用它成功培训了数千名专业的Android开发人员。本书将指导你完成多个Android应用的开发，并根据需要逐步介绍各种理论概念及技术知识。在学习过程中，如果遇到知识疑难点，请勇敢面对；我们也会尽最大努力抽丝剥茧，让你知其然更知其所以然。

我们的教学方法是：在学习理论的同时，就着手运用它们开发实际的应用，而非先学习一大堆理论，再考虑如何将理论应用于实践。读完本书，你将具备必要的开发经验及知识。以此为起点，深入学习，逐渐成长为一名合格的Android开发者。

阅读前提

使用本书，你需要熟悉Java语言，包括类、对象、接口、监听器、包、内部类、匿名内部类、泛型类等基本概念。

如果不熟悉这些概念，很可能刚翻几页就已无法继续下去。对此，建议先放下本书，找本Java入门书看一看。市面上有很多优秀的Java入门书，你可以基于自己的编程经验及学习风格去挑选。

如果你熟悉面向对象编程，但Java知识忘得差不多了，阅读本书应该不会有太大的问题。对于接口、匿名内部类等重要的Java语言点，我们会提供必要的简短回顾。建议在学习过程中手边备上一本Java参考书，方便查阅。

第3版有哪些新内容

本书第3版介绍了一些新工具：constraint layout（包括其编辑器）和数据绑定（data binding）。

新增加了几个章节，内容涉及单元测试、辅助功能（accessibility）、MVVM架构和应用本地化。本书末尾还介绍了Android的新运行时权限系统。此外，我们还进一步扩充了挑战练习和深入学习部分的内容，并修订了全书的一些不够完善的部分。

如何使用本书

本书基于Big Nerd Ranch培训机构的5天教学课程编写而成。课程从基础知识讲起，各章节内容以循序渐进的方式编排，建议不要跳读，以免学习效果大打折扣。显然，本书不适合作为参考书。本书旨在帮你跨越学习的初始障碍，进而充分利用其他各种参考资料和代码实例类图书来深入学习。

我们为学员提供了良好的培训环境：专门的培训教室、可口的美食、舒适的住宿条件、动力十足的学习伙伴，以及随时答疑解惑的指导老师。

你同样需要类似的良好环境。因此，应保证充足的睡眠，找一个安静的地方开始学习。参考以下建议也很有帮助：

(1) 组织朋友或同事组成兴趣小组学习；

(2) 集中安排时间逐章学习；

(3) 参与本书论坛的交流讨论（forums.bignerdranch.com）；

(4) 寻求Android开发高手的帮助。

本书内容

本书带你学习开发8个Android应用。有些应用很简单，一章即可讲完；有些则相对复杂。最复杂的一个应用跨越了13章。通过这些精心编排的应用，你能学到很多重要的理论知识和开发技巧，并获得最直接的开发经验。

- GeoQuiz
  本书的第一个应用，借此学习Android应用的基本组成、activity、界面布局以及显式intent。
- CriminalIntent
  本书中最复杂的应用，用来记录办公室同事的种种陋习。借此应用学习fragment、主从用户界面、list-backed用户界面、菜单选项、相机调用、隐式intent等内容。
- BeatBox
  一个可以震慑坏人的应用，借此深入学习fragment、媒体文件的播放与控制、MVVM架构、数据绑定、单元测试、主题以及drawable资源。
- NerdLauncher
  一个个性化启动器，借此深入学习intent以及Android任务。
- PhotoGallery
  一个从Flickr网站下载并显示照片的客户端应用，借此学习Android服务、多线程、网络内容下载等知识。

- DragAndDraw
  一个简单的画图应用，借此学习触摸手势事件处理以及如何创建个性化视图。
- Sunset
  一个漂亮的日落动画应用，借此学习Android动画。
- Locatr
  一个查询当前位置的Flickr图片并显示在地图上的应用。借此应用学习如何使用定位服务和地图。

挑战练习

大部分章末都配有练习题。可借此机会学以致用，查阅官方文档，锻炼独立解决问题的能力。

强烈建议你完成这些挑战练习。在练习过程中，尝试另辟蹊径，探索自己独特的学习之路。这有助于巩固所学知识，增强未来开发应用的信心。

若遇到一时难以解决的问题，请访问论坛http://forums.bignerdranch.com求助。

深入学习

部分章末还包含一块名为“深入学习”的内容。这些内容针对相应章节的知识点，提供深入讲解或更多学习信息。本部分内容不属于必须掌握的部分，但还是希望你有兴趣阅读并有所收获。

编码风格

有别于其他Android开发学习社区的编码风格，我们有自己的偏好，主要有以下两个方面。

- *在监听器代码部分使用匿名内部类*
  这纯属个人偏好。我们认为，使用匿名内部类，代码可以更简练，监听器实现方法更一目了然。尽管在高性能要求的场景下或大型应用中，匿名内部类可能会有一些问题，但大多数情况下没有问题。
- *自第7章引入fragment后，后续所有用户界面都使用它*
  我们有理由坚持这一点。相信我们，使用得当的话，fragment就是Android开发人员手中的一大利器。一旦适应，用起来也没那么难。相比activity，fragment在创建和显示用户界面时更加灵活，优势明显，值得为此付出努力。

版式说明

为方便阅读，本书会对某些特定内容采用专门的字体。变量、常量、类型、类名、接口名和方法名会以代码体显示。

所有代码与XML清单也会以代码体显示。需要输入的代码或XML总是以粗体显示。应该删

除的代码或XML会打上删除线。例如，在以下实现代码里，我们删除了makeText(...)方法的调用，增加了checkAnswer(true)方法的调用。

```
@Override
public void onClick(View v) {
    Toast.makeText(QuizActivity.this, R.string.incorrect_toast,
                   Toast.LENGTH_SHORT).show();
    checkAnswer(true);
}
```

Android 版本

本书教学主要针对当前广泛使用的各个系统版本（Android 4.4 至 Android 7.1）。虽然更老的系统版本仍有人在用，但对于大多数开发者来说，为这部分人开发应用就是个赔本的买卖。如果应用确实需要支持 Android 4.4 之前的系统版本，请参考本书第2版（Android 4.1及以上版本）和第1版（Android 2.3及以上版本）的相关内容。

Google还会不断地发布新版本的Android系统。请放心，Android支持向后兼容（详见第6章），即便有了新系统，本书所授知识也不会过时。而且，通过forums.bignerdranch.com论坛，我们也会不断跟踪Android开发新动向，及时为你提供开发指导和支持。

开发必备工具

开始学习前，你需要安装Android Studio。基于流行的IntelliJ IDEA创建，Android Studio是用于Android开发的一套集成开发工具。

Android Studio的安装包括如下内容。

- Android SDK
  最新版本的Android SDK。
- Android SDK工具和平台工具
  用来测试与调试应用的一套工具。
- Android模拟器系统镜像
  用来在不同虚拟设备上开发和测试应用。

撰写本书时，Google一直在积极开发和更新Android Studio版本。因此，请注意了解当前版本和本书所用版本之间的差异。如需帮助，请访问forums.bignerdranch.com论坛。

Android Studio的下载与安装

可以从Android开发者网站下载Android Studio：developer.android.com/sdk/。

首次安装的话，还需从www.oracle.com下载并安装Java开发工具箱（JDK 8）。

如仍有疑问，请访问网址developer.android.com/sdk/寻求帮助。

下载早期版本的SDK

Android Studio自带最新版本的SDK和模拟器系统镜像。若想在Android早期版本上测试应用，还需额外下载相关工具组件。

可通过Android SDK管理器来配置安装这些组件。在Android Studio中，选择Tools → Android → SDK Manager菜单项，如图0-1所示。（已创建并打开了新项目时，Tools菜单才可见。还没创建过项目的话，可在Android开发向导界面，在Quick Start区域，选择Configure → SDK Manager来启动SDK管理器。）

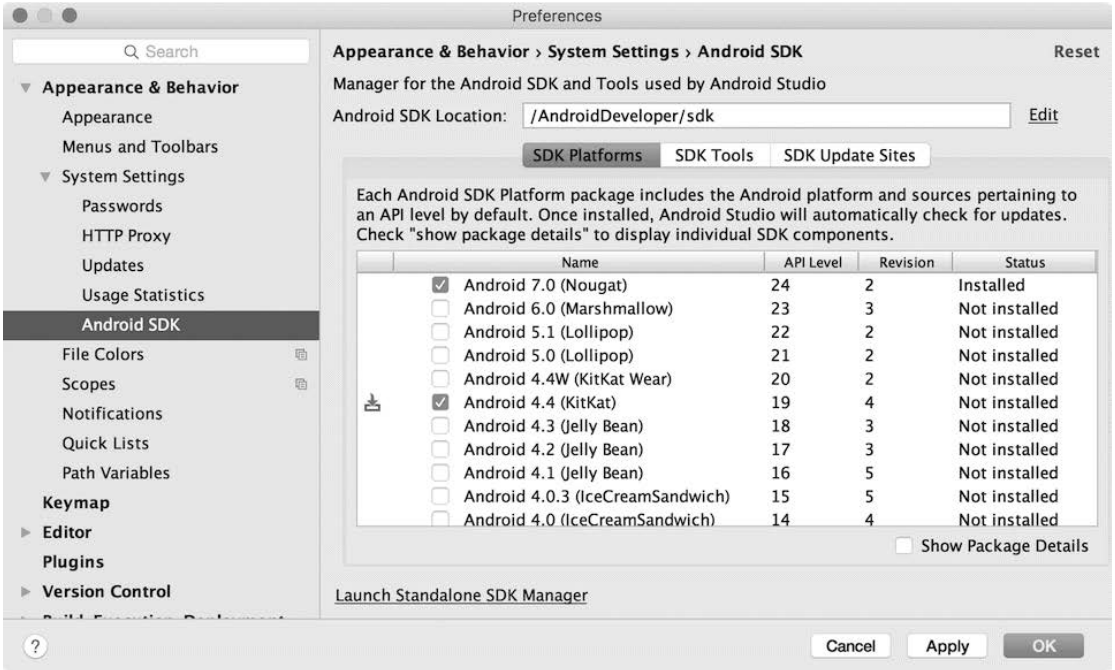


图0-1　Andriod SDK管理器

选择并安装需要的Android版本和工具。下载这些组件需要一定时间，请耐心等待。

通过Android SDK管理器，还可以及时获取Android最新发布内容，比如新系统平台或新版本工具等。

硬件设备

模拟器是测试应用的好帮手，但需测试应用性能时，物理Android设备无可替代。如果手头有物理设备，建议按需使用。

目　　录

第 1 章　Android 开发初体验 1
1.1　Android 开发基础 2
1.2　创建 Android 项目 2
1.3　Android Studio 使用导航 6
1.4　用户界面设计 6
1.4.1　视图层级结构 10
1.4.2　组件属性 10
1.4.3　创建字符串资源 11
1.4.4　预览布局 12
1.5　从布局 XML 到视图对象 13
1.6　组件的实际应用 16
1.6.1　引用组件 17
1.6.2　设置监听器 17
1.7　创建提示消息 19
1.8　使用模拟器运行应用 21
1.9　深入学习：Android 编译过程 23
1.10　关于挑战练习 25
1.11　挑战练习：定制 toast 消息 25
第 2 章　Android 与 MVC 设计模式 26
2.1　创建新类 26
2.2　Android 与 MVC 设计模式 29
2.3　更新视图层 31
2.4　更新控制器层 33
2.5　在物理设备上运行应用 37
2.5.1　连接设备 37
2.5.2　配置设备用于应用开发 37
2.6　添加图标资源 38
2.6.1　向项目中添加资源 39
2.6.2　在 XML 文件中引用资源 41
2.7　挑战练习：为 TextView 添加监听器 42
2.8　挑战练习：添加后退按钮 42
2.9　挑战练习：从按钮到图标按钮 42
第 3 章　activity 的生命周期 45
3.1　日志跟踪理解 activity 生命周期 46
3.1.1　输出日志信息 46
3.1.2　使用 LogCat 48
3.1.3　activity 生命周期实例解析 50
3.2　设备旋转与 activity 生命周期 52
3.3　保存数据以应对设备旋转 56
3.4　再探 activity 生命周期 58
3.5　深入学习：activity 内存清理现状 59
3.6　深入学习：日志记录的级别与方法 60
3.7　挑战练习：禁止一题多答 61
3.8　挑战练习：评分 61
第 4 章　Android 应用的调试 62
4.1　异常与栈跟踪 63
4.1.1　诊断应用异常 64
4.1.2　记录栈跟踪日志 65
4.1.3　设置断点 66
4.1.4　使用异常断点 68
4.2　Android 特有的调试工具 70
4.2.1　使用 Android Lint 70
4.2.2　R 类的问题 71
4.3　挑战练习：探索布局检查器 72
4.4　挑战练习：探索内存分配跟踪 72
第 5 章　第二个 activity 74
5.1　创建第二个 activity 75

5.1.1 创建新的 activity 75
5.1.2 创建新的 activity 子类 78
5.1.3 在 manifest 配置文件中声明 activity 78
5.1.4 为 QuizActivity 添加 CHEAT 按钮 79
5.2 启动 activity 81
5.3 activity 间的数据传递 83
5.3.1 使用 intent extra 83
5.3.2 从子 activity 获取返回结果 86
5.4 activity 的使用与管理 91
5.5 挑战练习：堵住漏洞 93

第 6 章 Android SDK 版本与兼容 94
6.1 Android SDK 版本 94
6.2 Android 编程与兼容性问题 95
6.2.1 比较合理的版本 95
6.2.2 SDK 最低版本 96
6.2.3 SDK 目标版本 97
6.2.4 SDK 编译版本 97
6.2.5 安全添加新版本 API 中的代码 97
6.3 使用 Android 开发者文档 100
6.4 挑战练习：报告编译版本 101
6.5 挑战练习：限制作弊次数 102

第 7 章 UI fragment 与 fragment 管理器 103
7.1 UI 设计的灵活性需求 104
7.2 引入 fragment 104
7.3 着手开发 CriminalIntent 105
7.3.1 创建新项目 107
7.3.2 两类 fragment 108
7.3.3 在 Android Studio 中增加依赖关系 108
7.3.4 创建 Crime 类 111
7.4 托管 UI fragment 112
7.4.1 fragment 的生命周期 112
7.4.2 托管的两种方式 113
7.4.3 定义容器视图 113
7.5 创建 UI fragment 115
7.5.1 定义 CrimeFragment 的布局 115
7.5.2 创建 CrimeFragment 类 117
7.6 向 FragmentManager 添加 UI fragment 122
7.6.1 fragment 事务 123
7.6.2 FragmentManager 与 fragment 生命周期 125
7.7 采用 fragment 的应用架构 126
7.8 深入学习：fragment 与支持库 127
7.9 深入学习：为什么优先使用支持库版 fragment 128

第 8 章 使用 RecyclerView 显示列表 129
8.1 升级 CriminalIntent 应用的模型层 130
8.2 使用抽象 activity 托管 fragment 132
8.2.1 通用型 fragment 托管布局 133
8.2.2 抽象 activity 类 133
8.3 RecyclerView、ViewHolder 和 Adapter 137
8.3.1 ViewHolder 和 Adapter 138
8.3.2 使用 RecyclerView 141
8.3.3 列表项视图 142
8.3.4 实现 ViewHolder 和 Adapter 143
8.4 绑定列表项 145
8.5 响应点击 147
8.6 深入学习：ListView 和 GridView 148
8.7 深入学习：单例 148
8.8 挑战练习：RecyclerView ViewType 149

第 9 章 使用布局与组件创建用户界面 150
9.1 使用图形布局工具 151
9.2 引入 ConstraintLayout 151
9.2.1 使用 ConstraintLayout 152
9.2.2 约束编辑器 153
9.2.3 腾出空间 155
9.2.4 添加组件 157
9.2.5 约束的 XML 形式 159
9.2.6 编辑属性 160
9.2.7 动态设置列表项 162

9.3 深入学习布局属性 …… 163
9.3.1 dp、sp 以及屏幕像素密度 …… 163
9.3.2 边距与内边距 …… 165
9.3.3 样式、主题及主题属性 …… 166
9.3.4 Android 应用的设计原则 …… 166
9.4 图形布局工具使用小结 …… 167
9.5 挑战练习：日期格式化 …… 167

第 10 章　使用 fragment argument …… 168

10.1 从 fragment 中启动 activity …… 168
10.1.1 附加 extra 信息 …… 169
10.1.2 获取 extra 信息 …… 170
10.1.3 使用 Crime 数据更新 CrimeFragment 视图 …… 171
10.1.4 直接获取 extra 信息的缺点 …… 172
10.2 fragment argument …… 172
10.2.1 附加 argument 给 fragment …… 173
10.2.2 获取 argument …… 174
10.3 刷新显示列表项 …… 174
10.4 通过 fragment 获取返回结果 …… 176
10.5 深入学习：为何要用 fragment argument …… 177
10.6 挑战练习：实现高效的 RecyclerView 刷新 …… 178
10.7 挑战练习：优化 CrimeLab 的表现 …… 178

第 11 章　使用 ViewPager …… 179

11.1 创建 CrimePagerActivity …… 180
11.1.1 ViewPager 与 PagerAdapter …… 181
11.1.2 整合并配置使用 CrimePagerActivity …… 182
11.2 FragmentStatePagerAdapter 与 FragmentPagerAdapter …… 184
11.3 深入学习：ViewPager 的工作原理 …… 186
11.4 深入学习：以代码的方式创建视图 …… 187
11.5 挑战练习：恢复 CrimeFragment 的边距 …… 187
11.6 挑战练习：添加 Jump to First 按钮和 Jump to Last 按钮 …… 188

第 12 章　对话框 …… 189

12.1 创建 DialogFragment …… 190
12.1.1 显示 DialogFragment …… 192
12.1.2 设置对话框的显示内容 …… 194
12.2 fragment 间的数据传递 …… 196
12.2.1 传递数据给 DatePickerFragment …… 197
12.2.2 返回数据给 CrimeFragment …… 199
12.3 挑战练习：更多对话框 …… 205
12.4 挑战练习：实现响应式 DialogFragment …… 205

第 13 章　工具栏 …… 207

13.1 AppCompat …… 207
13.2 工具栏菜单 …… 209
13.2.1 在 XML 文件中定义菜单 …… 210
13.2.2 创建菜单 …… 214
13.2.3 响应菜单项选择 …… 217
13.3 实现层级式导航 …… 218
13.4 可选菜单项 …… 219
13.4.1 切换菜单项标题 …… 220
13.4.2 “还有个问题” …… 222
13.5 深入学习：工具栏与操作栏 …… 224
13.6 挑战练习：删除 crime 记录 …… 224
13.7 挑战练习：复数字符串资源 …… 224
13.8 挑战练习：用于 RecyclerView 的空视图 …… 225

第 14 章　SQLite 数据库 …… 226

14.1 定义 schema …… 226
14.2 创建初始数据库 …… 227
14.2.1 使用 Android Device Monitor 查看文件 …… 230
14.2.2 处理数据库相关问题 …… 231
14.3 修改 CrimeLab 类 …… 232
14.4 写入数据库 …… 233
14.4.1 使用 ContentValues …… 233
14.4.2 插入和更新记录 …… 234
14.5 读取数据库 …… 235
14.5.1 使用 CursorWrapper …… 236

14.5.2 创建模型层对象 238
14.6 深入学习：数据库高级主题介绍 241
14.7 深入学习：应用上下文 241
14.8 挑战练习：删除 crime 记录 242

第 15 章 隐式 intent 243
15.1 添加按钮组件 244
15.2 添加嫌疑人信息至模型层 244
15.3 使用格式化字符串 246
15.4 使用隐式 intent 248
15.4.1 隐式 intent 的组成 248
15.4.2 发送消息 249
15.4.3 获取联系人信息 252
15.4.4 检查可响应任务的 activity 254
15.5 挑战练习：ShareCompat 256
15.6 挑战练习：又一个隐式 intent 256

第 16 章 使用 intent 拍照 258
16.1 布置照片 258
16.2 文件存储 260
16.2.1 使用 FileProvider 261
16.2.2 指定照片存放位置 262
16.3 使用相机 intent 263
16.4 缩放和显示位图 265
16.5 功能声明 268
16.6 挑战练习：优化照片显示 269
16.7 挑战练习：优化缩略图加载 269

第 17 章 双版面主从用户界面 270
17.1 增加布局灵活性 271
17.1.1 修改 SingleFragment-Activity 272
17.1.2 创建包含两个 fragment 容器的布局 272
17.1.3 使用别名资源 274
17.1.4 创建平板设备专用可选资源 275
17.2 activity：fragment 的托管者 276
17.3 深入学习：设备屏幕尺寸的确定 284
17.4 挑战练习：添加滑动删除功能 285

第 18 章 应用本地化 286
18.1 资源本地化 286
18.1.1 默认资源 289
18.1.2 检查资源本地化完成情况 291
18.1.3 区域修饰符 292
18.2 配置修饰符 294
18.2.1 可用资源优先级排定 295
18.2.2 多重配置修饰符 296
18.2.3 寻找最匹配的资源 297
18.3 测试备选资源 298
18.4 挑战练习：日期本地化 298

第 19 章 Android 辅助功能 299
19.1 TalkBack 299
19.1.1 点击浏览 301
19.1.2 线性浏览 301
19.2 实现非文字型元素可读 303
19.2.1 添加内容描述 303
19.2.2 实现组件可聚焦 305
19.3 提升辅助体验 305
19.4 深入学习：使用辅助功能扫描器 309
19.5 挑战练习：优化列表项 311
19.6 挑战练习：补全上下文信息 311
19.7 挑战练习：事件主动通知 311

第 20 章 数据绑定与 MVVM 313
20.1 为何要用 MVVM 架构 314
20.2 创建 BeatBox 应用 314
20.3 导入 assets 319
20.4 处理 assets 321
20.5 使用 assets 323
20.6 绑定数据 325
20.6.1 创建视图模型 327
20.6.2 绑定至视图模型 328
20.6.3 绑定数据观察 330
20.7 访问 assets 332
20.8 深入学习：数据绑定再探 333
20.8.1 lambda 表达式 333
20.8.2 更多语法糖 333
20.8.3 BindingAdapter 333

20.9　深入学习：为何使用 assets……334
20.10　深入学习：什么是 non-assets……334

第 21 章　音频播放与单元测试……335

21.1　创建 SoundPool……335
21.2　加载音频文件……336
21.3　播放音频……337
21.4　添加测试依赖……338
21.5　创建测试类……339
21.6　实现测试类……341
21.7　编写测试方法……342
21.8　数据绑定回调……346
21.9　释放音频……346
21.10　设备旋转和对象保存……347
21.10.1　保留 fragment……348
21.10.2　设备旋转和已保留的 fragment……349
21.11　深入学习：是否保留 fragment……351
21.12　深入学习：Espresso 与整合测试……351
21.13　深入学习：虚拟对象与测试……352
21.14　挑战练习：播放进度控制……353

第 22 章　样式与主题……354

22.1　颜色资源……354
22.2　样式……355
22.3　主题……357
22.4　添加主题颜色……359
22.5　覆盖主题属性……361
22.6　修改按钮属性……364
22.7　深入学习：样式继承拾遗……367
22.8　深入学习：引用主题属性……367

第 23 章　XML drawable……368

23.1　统一按钮样式……368
23.2　shape drawable……369
23.3　state list drawable……371
23.4　layer list drawable……372
23.5　深入学习：为什么要用 XML drawable……373
23.6　深入学习：使用 mipmap 图像……374
23.7　深入学习：使用 9-patch 图像……375
23.8　挑战练习：按钮主题……379

第 24 章　深入学习 intent 和任务……380

24.1　创建 NerdLauncher 项目……380
24.2　解析隐式 intent……382
24.3　在运行时创建显式 intent……386
24.4　任务与回退栈……388
24.4.1　在任务间切换……389
24.4.2　启动新任务……389
24.5　使用 NerdLauncher 应用作为设备主屏幕……392
24.6　挑战练习：应用图标……393
24.7　深入学习：进程与任务……393
24.8　深入学习：并发文档……395

第 25 章　HTTP 与后台任务……398

25.1　创建 PhotoGallery 应用……399
25.2　网络连接基本……401
25.3　使用 AsyncTask 在后台线程上运行代码……403
25.4　线程与主线程……405
25.5　从 Flickr 获取 JSON 数据……407
25.6　从 AsyncTask 回到主线程……413
25.7　清理 AsyncTask……416
25.8　深入学习：AsyncTask 再探……416
25.9　深入学习：AsyncTask 的替代方案……417
25.10　挑战练习：Gson……418
25.11　挑战练习：分页……418
25.12　挑战练习：动态调整网格列……418

第 26 章　Looper、Handler 和 HandlerThread……420

26.1　配置 RecyclerView 以显示图片……420
26.2　批量下载缩略图……422
26.3　与主线程通信……423
26.4　创建并启动后台线程……424
26.5　Message 与 message handler……426
26.5.1　剖析 Message……426
26.5.2　剖析 Handler……427
26.5.3　使用 handler……428
26.5.4　传递 handler……431

26.6 深入学习：AsyncTask 与线程 ······436
26.7 深入学习：解决图片下载问题 ······436
26.8 深入学习：StrictMode ······437
26.9 挑战练习：预加载以及缓存 ······438

第 27 章 搜索 ······439

27.1 搜索 Flickr 网站 ······439
27.2 使用 SearchView ······443
27.3 使用 shared preferences 实现轻量级数据存储 ······447
27.4 优化应用 ······451
27.5 挑战练习：深度优化 PhotoGallery 应用 ······451

第 28 章 后台服务 ······452

28.1 创建 IntentService ······452
28.2 服务的作用 ······454
28.3 查找最新返回结果 ······456
28.4 使用 AlarmManager 延迟运行服务 ······458
28.4.1 合理控制服务启动的频度 ······460
28.4.2 PendingIntent ······461
28.4.3 使用 PendingIntent 管理定时器 ······461
28.5 控制定时器 ······462
28.6 通知信息 ······464
28.7 挑战练习：可穿戴设备上的通知 ······466
28.8 深入学习：服务之细节 ······467
28.8.1 服务的能与不能 ······467
28.8.2 服务的生命周期 ······467
28.8.3 non-sticky 服务 ······467
28.8.4 sticky 服务 ······468
28.8.5 绑定服务 ······468
28.9 深入学习：JobScheduler 和 JobService ······469
28.10 挑战练习：在 Lollipop 设备上使用 JobService ······472
28.11 深入学习：sync adapter ······472

第 29 章 broadcast intent ······474

29.1 普通 intent 和 broadcast intent ······474
29.2 接收系统 broadcast：重启后唤醒 ······475
29.2.1 创建并登记 standalone receiver ······475
29.2.2 使用 receiver ······477
29.3 过滤前台通知消息 ······479
29.3.1 发送 broadcast intent ······479
29.3.2 创建并登记动态 receiver ······480
29.3.3 使用私有权限限制 broadcast ······482
29.3.4 使用有序 broadcast 收发数据 ······484
29.4 receiver 与长时运行任务 ······488
29.5 深入学习：本地事件 ······488
29.5.1 使用 EventBus ······489
29.5.2 使用 RxJava ······490
29.6 深入学习：探测 fragment 的状态 ······490

第 30 章 网页浏览 ······492

30.1 最后一段 Flickr 数据 ······492
30.2 简单方式：隐式 intent ······494
30.3 较难方式：使用 WebView ······496
30.4 处理 WebView 的设备旋转问题 ······502
30.5 深入学习：注入 JavaScript 对象 ······502
30.6 深入学习：WebView 升级 ······503
30.7 挑战练习：使用后退键浏览历史网页 ······504
30.8 挑战练习：非 HTTP 链接支持 ······504

第 31 章 定制视图与触摸事件 ······505

31.1 创建 DragAndDraw 项目 ······505
31.2 创建定制视图 ······507
31.3 处理触摸事件 ······509
31.4 onDraw(...)方法内的图形绘制 ······513
31.5 挑战练习：设备旋转问题 ······515
31.6 挑战练习：旋转矩形框 ······515

第 32 章 属性动画 ······516

32.1 建立场景 ······516
32.2 简单属性动画 ······519
32.2.1 视图属性转换 ······521

32.2.2 使用不同的 interpolator ········523

32.2.3 色彩渐变 ········523

32.3 播放多个动画 ········525

32.4 深入学习：其他动画 API ········527

32.4.1 传统动画工具 ········527

32.4.2 转场 ········527

32.5 挑战练习 ········527

第 33 章 地理位置和 Play 服务 ········528

33.1 地理位置和定位类库 ········528

33.2 创建 Locatr 项目 ········529

33.3 Play 服务定位和模拟器 ········530

33.4 创建 Locatr 应用 ········532

33.5 配置 Google Play 服务 ········534

33.6 使用 Google Play 服务 ········536

33.7 基于地理位置的 Flickr 搜索 ········538

33.8 获取定位数据 ········539

33.9 获取运行时权限 ········542

33.10 寻找并显示图片 ········546

33.11 挑战练习：权限使用理由 ········548

33.12 挑战练习：进度指示器 ········549

第 34 章 使用地图 ········550

34.1 导入 Play 地图服务库 ········550

34.2 Android 上的地图服务 ········550

34.3 获取 Maps API key ········550

34.4 创建地图 ········552

34.5 获取更多地理位置数据 ········553

34.6 使用地图 ········556

34.7 深入学习：团队开发和 API key ········560

第 35 章 material design ········562

35.1 material surface ········562

35.1.1 elevation 和 Z 值 ········564

35.1.2 state list animator ········565

35.2 动画工具 ········566

35.2.1 circular reveal ········567

35.2.2 shared element transition ········568

35.3 新的视图组件 ········571

35.3.1 card ········571

35.3.2 floating action button ········572

35.3.3 snackbar ········573

35.4 深入学习 material design ········574

第 36 章 编后语 ········575

36.1 终极挑战 ········575

36.2 关于我们 ········575

36.3 致谢 ········576

第 1 章 Android开发初体验

本章将带你开发本书第一个应用，并借此学习一些Android基本概念以及构成应用的UI组件。学完本章，如果没能全部理解，也不必担心，后续章节还会涉及这些内容并有更加详细的讲解。

马上要开发的应用名叫GeoQuiz，它能给出一道道地理知识问题。用户点击TRUE或FALSE按钮来回答屏幕上的问题，GeoQuiz即时作出反馈。

图1-1显示了用户点击TRUE按钮的结果。

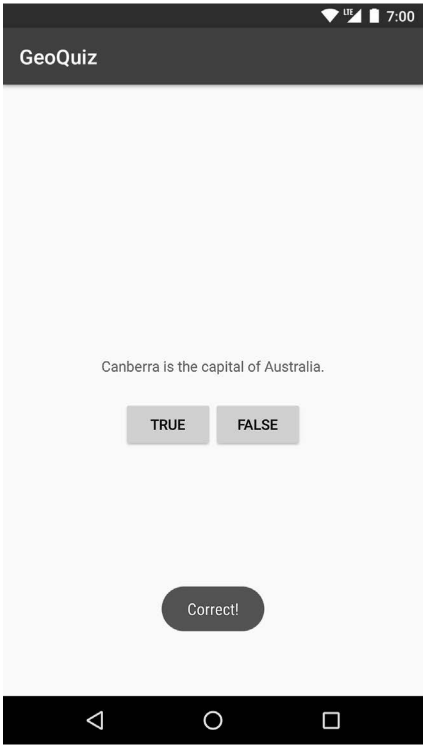


图1-1　你是澳洲人吗

1.1　Android 开发基础

GeoQuiz应用由一个activity和一个布局（layout）组成。

- activity是Android SDK中`Activity`类的一个实例，负责管理用户与应用界面的交互。应用的功能是通过编写`Activity`子类来实现的。对于简单的应用来说，一个`Activity`子类可能就够了，而复杂的应用则会有多个。
  GeoQuiz是个简单应用，因此它只有一个名叫`QuizActivity`的`Activity`子类。`QuizActivity`管理着图1-1所示的用户界面。
- 布局定义了一系列用户界面对象以及它们显示在屏幕上的位置。组成布局的定义保存在XML文件中。每个定义用来创建屏幕上的一个对象，如按钮或文本信息。
  GeoQuiz应用包含一个名叫activity_quiz.xml的布局文件。该布局文件中的XML标签定义了图1-1所示的用户界面。

`QuizActivity`与activity_quiz.xml文件的关系如图1-2所示。

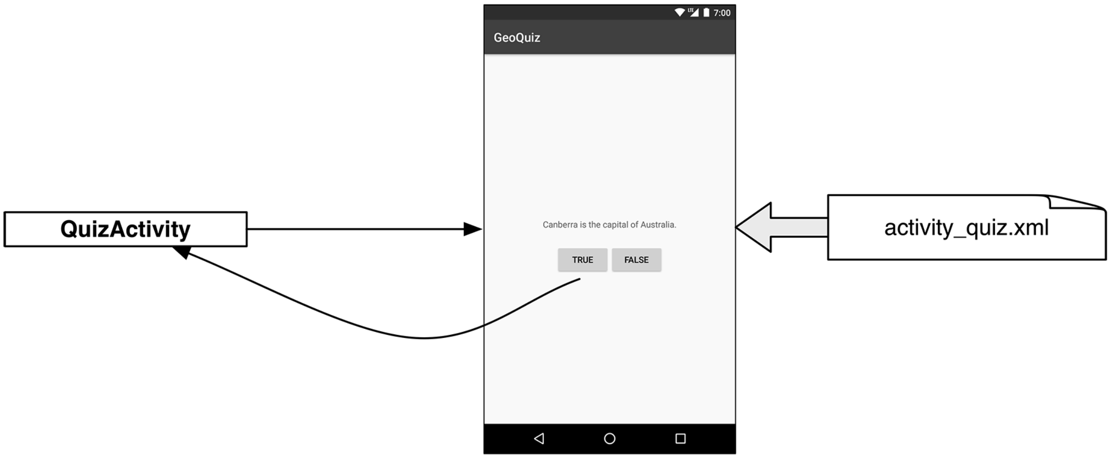


图1-2　`QuizActivity`管理着activity_quiz.xml文件定义的用户界面

有了这些Android基本概念之后，我们来创建GeoQuiz应用。

1.2　创建 Android 项目

首先我们创建一个Android项目。Android项目包含组成一个应用的全部文件。

启动Android Studio程序，首次运行的话，会看到如图1-3所示的欢迎界面。



图1-3　欢迎使用Android Studio

在欢迎界面，选择创建Android Studio新项目选项（Start a new Android Studio project）；非首次运行的话，选择File → New → New Project…菜单项。

现在，你应该打开了新建项目向导界面，如图1-4所示。在此界面的应用名称（Application name）处输入GeoQuiz。在公司域名（Company Domain）处输入android.bignerdranch.com。此时自动产生的包名称（Package name）会变为com.bignerdranch.android.geoquiz。至于项目存储位置（Project location），就看个人喜好了。

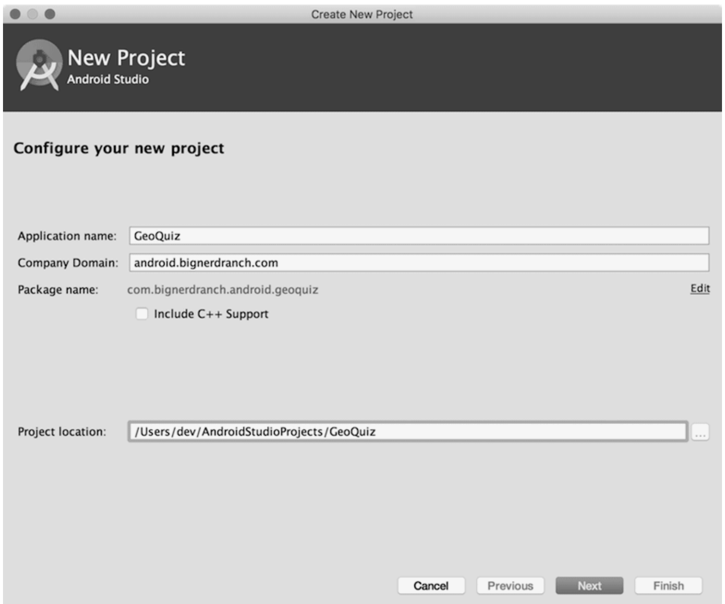


图1-4　创建新项目

注意，以上包名称遵循了“DNS反转”约定，也就是将企业组织或公司的域名反转后，在尾部附加上应用名称。遵循此约定可以保证包名称的唯一性，这样，同一设备和Google Play商店的各类应用就可以区分开来。

单击Next按钮，接下来配置应用支持哪些版本的Android设备。GeoQuiz应用只能在手机上运行，所以这里勾选Phone and Tablet选项。SDK最低版本选择API 19: Android 4.4 (KitKat)，如图1-5所示。第6章会介绍Android不同SDK版本的差异。



图1-5　设备支持配置

单击Next按钮继续。

在接下来的窗口中，需要为GeoQuiz应用的启动初始屏选择模板，如图1-6所示。选择Empty Activity后单击Next按钮继续。

（Android Studio更新频繁，因此新版本的向导界面可能与本书略有不同。这不是什么大问题，一般来讲，工具更新后，向导界面的配置选项应该不会有太大差别。如果大有不同，说明开发工具有了重大更新。不要担心，请访问本书论坛 forums.bignerdranch.com，我们会教你如何使用新版本的开发工具。）

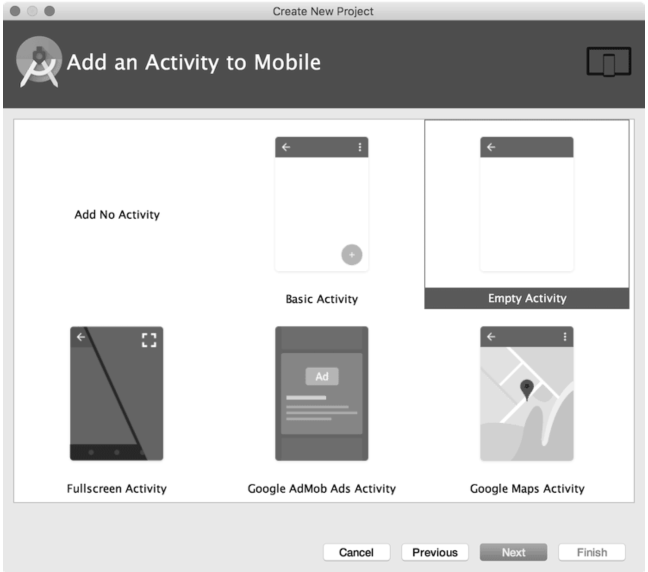


图1-6　选取activity类型（空activity）

在应用向导的最后一个窗口，命名activity子类为`QuizActivity`，如图1-7所示。注意子类名的`Activity`后缀。尽管不是必需的，但最好是遵循这种规范的命名约定。

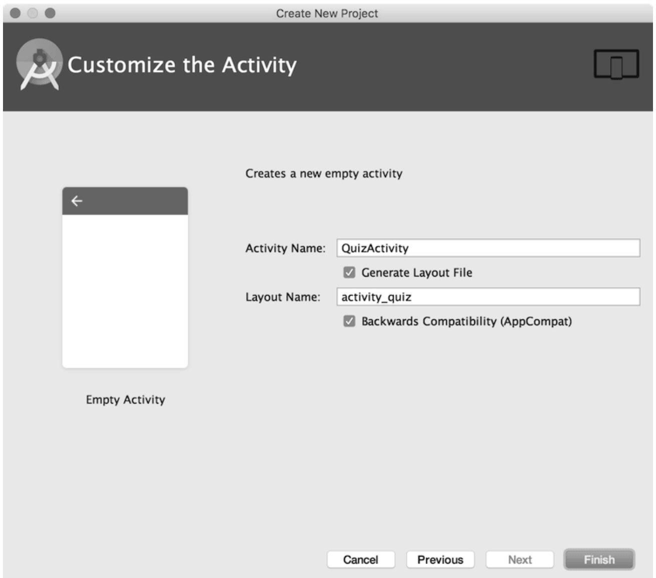


图1-7　配置activity

确认Generate Layout File选项已勾选。为体现布局与activity间的对应关系，布局名（Layout

Name）会自动更新为`activity_quiz`。布局的命名规则是：颠倒activity子类名的单词顺序并全部转小写，然后在单词间添加下划线。对于后续章节中的所有布局以及将要学习的其他资源，都建议采用这种命名风格。

如果你的Android Studio版本还有其他选项，请保持默认选择不变。单击Finish按钮，Android Studio会完成创建并打开新项目。

1.3　Android Studio 使用导航

如图1-8所示，Android Studio已在工作区窗口里打开新建项目。

整个工作区窗口分为不同的区域，这里统称为*工具窗口*（tool window）。

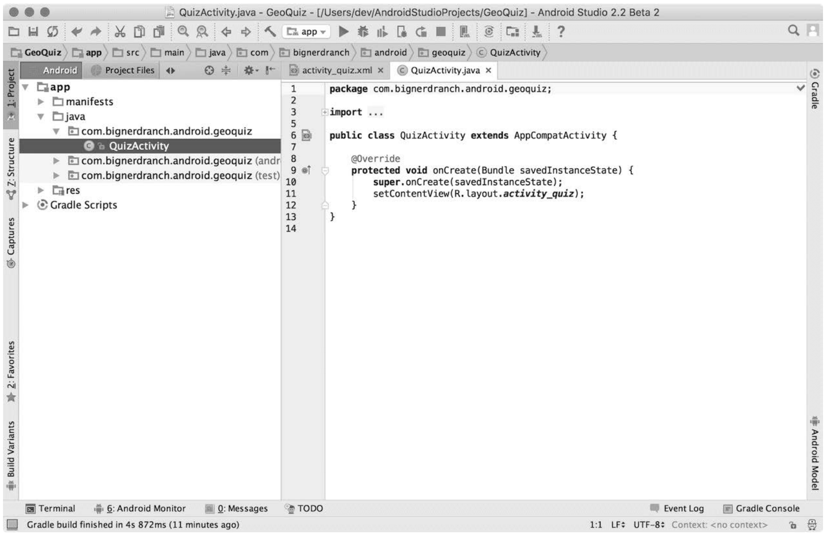


图1-8　新的项目窗口

左边是*项目工具窗口*（project tool window），通过它可以查看和管理所有与项目相关的文件。

主视图是代码编辑区（editor）。为便于开发，Android Studio已在代码编辑区打开了QuizActivity.java文件。

点击工作区窗口左边、右边以及底部标有各种名称的工具按钮区域，可显示或隐藏各类工具窗口。当然，也可以直接使用它们对应的快捷键。假如看不到某个工具按钮的话，可以点击左下角的灰色方形区域或单击View → Tool Buttons菜单项找到它。

1.4　用户界面设计

首先打开app/res/layout/activity_quiz.xml文件。如果看到的是布局预览界面，请点击底部的Text页切换显示XML代码。

当前，activity_quiz.xml文件定义了默认的activity布局。默认的XML布局文件内容经常有变，但相比代码清单1-1，一般不会有很大出入。

代码清单1-1　默认的activity布局（activity_quiz.xml）

```
<Relativelayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_quiz"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingleft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.bignerdranch.android.geoquiz.QuizActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/>
</Relativelayout>
```

应用activity的默认布局定义了两个组件（widget）：`RelativeLayout`和`TextView`。

组件是用户界面的构造模块。组件可以显示文字或图像，与用户交互，甚至布置屏幕上的其他组件。按钮、文本输入控件和选择框等都是组件。

Android SDK内置了多种组件，通过配置各种组件可获得所需的用户界面及行为。每一个组件都是`View`类或其子类（如`TextView`或`Button`）的一个具体实例。

图1-9展示了代码清单1-1中定义的`RelativeLayout`和`TextView`是如何在屏幕上显示的。

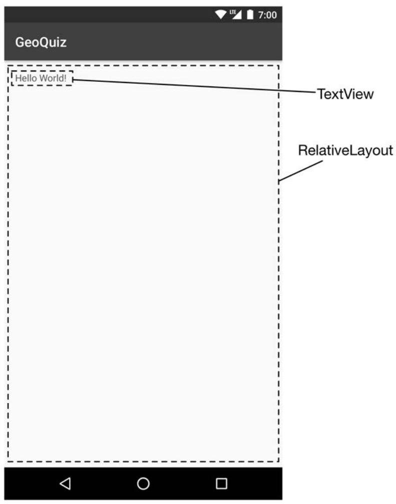


图1-9　显示在屏幕上的默认组件

不过，图1-9所示的默认组件并不是我们需要的，QuizActivity的用户界面需要下列组件：

- 一个垂直LinearLayout组件；
- 一个TextView组件；
- 一个水平LinearLayout组件；
- 两个Button组件。

图1-10展示了以上组件是如何构成QuizActivity用户界面的。

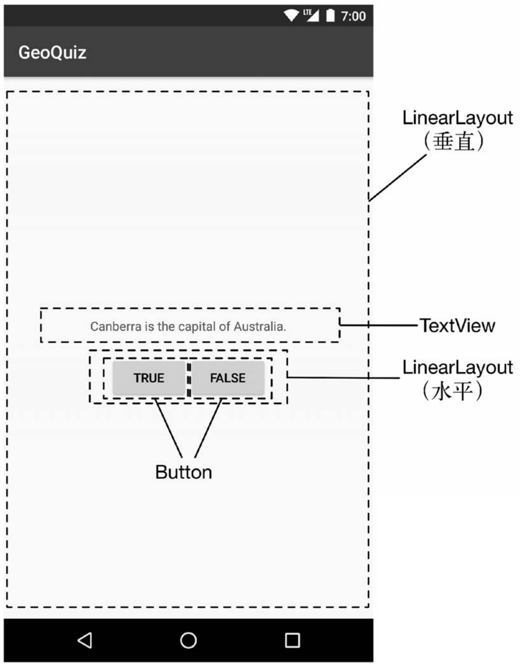


图1-10　布置并显示在屏幕上的组件

下面我们在activity_quiz.xml文件中定义这些组件。

在项目工具窗口中找到app/res/layout目录，打开activity_quiz.xml文件。对照代码清单1-2，修改文件内容。注意，需删除的XML已打上删除线，需添加的XML以粗体显示。本书统一使用这样的代码增删处理模式。

代码清单1-2　在XML文件（activity_quiz.xml）中定义组件

```
<RelativeLayout
    android:id="@+id/activity_quiz"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.bignerdranch.android.geoquiz.QuizActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/>
</RelativeLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/true_button" />

      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```

参照代码清单输入代码，暂时不理解这些代码也没关系，你会在后续学习中逐渐弄明白的。注意，开发工具无法校验布局XML内容，拼写错误早晚会出问题，应尽量避免。

可以看到，有三行以`android:text`开头的代码出现了错误信息。暂时忽略它们，稍后会处理。

对照图1-10所示的用户界面查看XML文件，可以看出组件与XML元素一一对应。元素的名称就是组件的类型。

各元素均有一组XML属性。*属性*可以看作如何配置组件的指令。

为方便理解元素与属性的工作原理，接下来我们将以层级视角来研究布局。

1.4.1 视图层级结构

组件包含在视图（View）对象的层级结构中，这种结构又称作视图层级结构（view hierarchy）。图1-11展示了代码清单1-2所示的XML布局对应的视图层级结构。

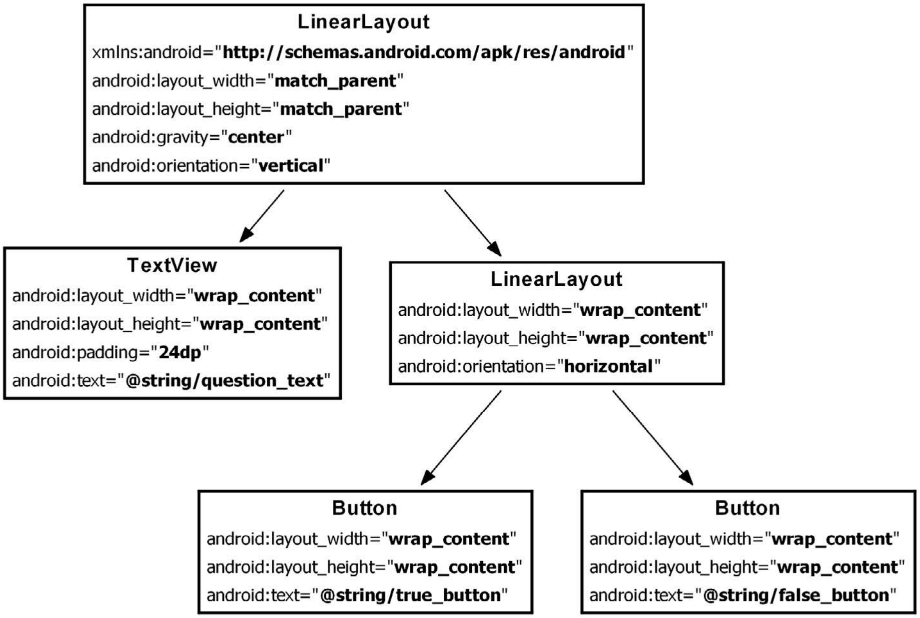


图1-11 布局组件的层级结构

从布局的视图层级结构可以看到，其根元素是一个LinearLayout组件。作为根元素，LinearLayout组件必须指定Android XML资源文件的命名空间属性，这里是http://schemas.android.com/apk/res/android。

LinearLayout组件继承自ViewGroup组件（也是个View子类）。ViewGroup组件是包含并配置其他组件的特殊组件。想要以一列或一排的样式布置组件，就可以使用LinearLayout组件。其他ViewGroup子类还有FrameLayout、TableLayout和RelativeLayout。

若某个组件包含在一个ViewGroup中，该组件与ViewGroup即构成父子关系。根LinearLayout有两个子组件：TextView和另一个LinearLayout。作为子组件的LinearLayout自己还有两个Button子组件。

1.4.2 组件属性

下面来看看配置组件时常用的一些属性。

1. android:layout_width和android:layout_height属性

几乎每类组件都需要android:layout_width和android:layout_height属性。以下是它

们的两个常见属性值（二选一）。

- match_parent：视图与其父视图大小相同。
- wrap_content：视图将根据其显示内容自动调整大小。

（以前还有个fill_parent属性值，等同于match_parent，现已废弃不用。）

根LinearLayout组件的高度与宽度属性值均为match_parent。LinearLayout虽然是根元素，但它也有父视图——Android提供该父视图来容纳应用的整个视图层级结构。

其他包含在界面布局中的组件，其高度与宽度属性值均被设置为wrap_content。请参照图1-10理解该属性值定义尺寸大小的作用。

TextView组件比其包含的文字内容区域稍大一些，这主要是android:padding="24dp"属性的作用。该属性告诉组件在决定大小时，除内容本身外，还需增加额外指定量的空间。这样屏幕上显示的问题与按钮之间便会留有一定的空间，使整体显得更为美观。（不理解dp的意思？dp即density-independent pixel，指与密度无关的像素，详见第9章。）

2. android:orientation属性

android:orientation属性是两个LinearLayout组件都具有的属性，它决定两者的子组件是水平放置还是垂直放置。根LinearLayout是垂直的，子LinearLayout是水平的。

子组件的定义顺序决定其在屏幕上显示的顺序。在垂直的LinearLayout中，第一个定义的子组件出现在屏幕的最上端；而在水平的LinearLayout中，第一个定义的子组件出现在屏幕的最左端。（如果设备文字从右至左显示，如阿拉伯语或者希伯来语，第一个定义的子组件则出现在屏幕的最右端。）

3. android:text属性

TextView与Button组件具有android:text属性。该属性指定组件要显示的文字内容。

请注意，android:text属性值不是字符串值，而是对字符串资源（string resource）的引用。

字符串资源包含在一个独立的名叫strings的XML文件中（strings.xml），虽然可以硬编码设置组件的文本属性值，如android:text="True"，但这通常不是个好主意。比较好的做法是：将文字内容放置在独立的字符串资源XML文件中，然后引用它们。这样会方便应用的本地化（支持多国语言）。

需要在activity_quiz.xml文件中引用的字符串资源还没添加。现在就来处理。

1.4.3　创建字符串资源

每个项目都包含一个默认字符串资源文件strings.xml。

在项目工具窗口中，找到app/res/values目录，展开目录，打开strings.xml文件。

可以看到，项目模板已经添加了一些字符串资源。如代码清单1-3所示，添加应用布局需要的三个新的字符串。

代码清单1-3　添加字符串资源（strings.xml）

```
<resources>
```

```
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
</resources>
```

（某些版本的Android Studio的strings.xml默认带有其他字符串，请勿删除它们，否则会引发与其他文件的联动错误。）

现在，在GeoQuiz项目的任何XML文件中，只要引用到`@string/false_button`，应用运行时，就会得到文本“False”。

保存strings.xml文件。这时，activity_quiz.xml布局缺少字符串资源的提示信息应该就消失了。（如仍有错误提示，请检查一下这两个文件，确认没有拼写错误。）

默认的字符串文件虽然已命名为strings.xml，你仍可以按个人喜好重命名。一个项目也可以有多个字符串文件。只要这些文件都放在res/values/目录下，含有一个`resources`根元素，以及多个`string`子元素，应用就能找到并正确使用它们。

1.4.4　预览布局

至此，应用的界面布局已经完成，可以使用图形布局工具实时预览了。首先，确认保存了所有相关文件并且无错误发生，然后回到activity_quiz.xml文件，点击代码编辑区右边的Preview打开预览工具窗口（如果还没打开的话），如图1-12所示。



图1-12　在图形布局工具中预览布局（activity_quiz.xml）

1.5 从布局 XML 到视图对象

知道activity_quiz.xml中的XML元素是如何转换为视图对象的吗？答案就在于QuizActivity类。

在创建GeoQuiz项目的同时，向导也创建了一个名叫QuizActivity的Activity子类。QuizActivity类文件存放在项目的app/java目录下。java目录是项目全部Java源代码的存放处。

在项目工具窗口中，依次展开app/java目录与com.bignerdranch.android.geoquiz包。找到并打开QuizActivity.java文件，查看其中的代码，如代码清单1-4所示。

代码清单1-4 默认QuizActivity类文件（QuizActivity.java）

```
package com.bignerdranch.android.geoquiz;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class QuizActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```

（是不是不明白AppCompatActivity的作用？它实际就是一个Activity子类，能为Android旧版本系统提供兼容支持。第13章会详细介绍AppCompatActivity。）

如果无法看到全部类包导入语句，请单击第一行导入语句左边的⊕符号来显示它们。

该Java类文件有一个Activity方法：onCreate(Bundle)。

（如果你的文件还包含onCreateOptionsMenu(Menu)和onOptionsItemSelected(MenuItem)方法，暂时不用理会。第13章会详细介绍它们。）

activity子类的实例创建后，onCreate(Bundle)方法会被调用。activity创建后，它需要获取并管理用户界面。要获取activity的用户界面，可调用以下Activity方法：

```
public void setContentView(int layoutResID)
```

根据传入的布局资源ID参数，该方法生成指定布局的视图并将其放置在屏幕上。布局视图生成后，布局文件包含的组件也随之以各自的属性定义完成实例化。

资源与资源 ID

布局是一种资源。资源是应用非代码形式的内容，如图像文件、音频文件以及XML文件等。

项目的所有资源文件都存放在目录app/res的子目录下。在项目工具窗口中可以看到，activity_quiz.xml布局资源文件存放在res/layout/目录下。strings.xml字符串资源文件存放在res/values/目录下。

可以使用资源ID在代码中获取相应的资源。activity_quiz.xml布局的资源ID为R.layout.activity_quiz。

查看GeoQuiz应用的资源ID需要切换项目视图。Android Studio默认使用Android项目视图，如图1-13所示。为让开发者专注于最常用的文件和目录，默认视图隐藏了Android项目的真实文件目录结构。

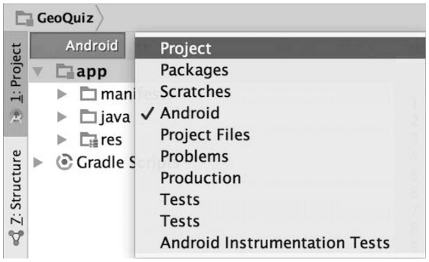


图1-13　切换项目视图

在项目工具窗口的最上部找到下拉菜单，从Android项目视图切换至Project视图。Project视图会显示出当前项目的所有文件和目录。

展开目录app/build/generated/source/r/debug，找到项目包名称并打开其中的R.java文件，即可看到GeoQuiz应用当前所有的资源。R.java文件在Android项目编译过程中自动生成，如该文件头部的警示所述，请不要修改该文件的内容。

修改布局或字符串等资源后，R.java文件不会实时刷新。Android Studio另外还存有一份代码编译用的R.java隐藏文件。当前代码编辑区打开的R.java文件仅在应用安装至设备或模拟器前产生，因此只有在Android Studio中点击运行应用时，它才会得到更新。

R.java文件通常比较大，代码清单1-5仅展示了部分内容。

代码清单1-5　GeoQuiz应用当前的资源ID（R.java）

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package com.bignerdranch.android.geoquiz;

public final class R {
    public static final class anim {
        ...
    }

    ...

    public static final class id {
```

```
            ...
        }
        public static final class layout {
            ...
            public static final int activity_quiz=0x7f030017;
        }
        public static final class mipmap {
            public static final int ic_launcher=0x7f030000;
        }
        public static final class string {
            ...
            public static final int app_name=0x7f0a0010;
            public static final int false_button=0x7f0a0012;
            public static final int question_text=0x7f0a0014;
            public static final int true_button=0x7f0a0015;
        }
    }
```

可以看到R.layout.activity_quiz即来自该文件。activity_quiz是R的内部类layout里的一个整型常量名。

GeoQuiz应用需要的字符串同样具有资源ID。目前为止，我们还未在代码中引用过字符串，如果需要，可以使用以下方法：

```
setTitle(R.string.app_name);
```

Android为整个布局文件以及各个字符串生成资源ID，但activity_quiz.xml布局文件中的组件除外，因为不是所有组件都需要资源ID。在本章中，我们要在代码里与两个按钮交互，因此只需为它们生成资源ID即可。

本书主要使用Android项目视图，生成资源ID之前，记得切回。当然，如果你就喜欢使用Project视图，也没啥问题。

要为组件生成资源ID，请在定义组件时为其添加android:id属性。在activity_quiz.xml文件中，分别为两个按钮添加上android:id属性，如代码清单1-6所示。

代码清单1-6　为按钮添加资源ID（activity_quiz.xml）

```
<LinearLayout  ... >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/true_button"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button" />

        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```

注意，android:id属性值前面有一个+标志，而android:text属性值则没有。这是因为我们在创建资源ID，而对字符串资源只是做引用。

1.6　组件的实际应用

按钮有了资源ID，就可以在QuizActivity中直接获取它们。首先，在QuizActivity.java文件中增加两个成员变量。

在QuizActivity.java文件中输入代码清单1-7所示代码。（请勿使用代码自动补全功能。）

代码清单1-7　添加成员变量（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
}
```

文件保存后，会看到两个错误提示。没关系，稍后会处理。请注意新增的两个成员（实例）变量名称的m前缀。该前缀是Android编程应遵循的命名约定，本书将始终遵循该约定。

现在，将鼠标移至代码左边的错误提示处时，会看到两条同样的错误信息：Cannot resolve symbol 'Button'。

这告诉我们，需要在QuizActivity.java文件中导入android.widget.Button类包。可在文件头部手动输入以下代码：

```
import android.widget.Button;
```

或者使用Option+Return（Alt+Enter）组合键，让Android Studio自动为你导入。代码有误时，也可以使用该组合键来修正。记得要常用。

类包导入后，刚才的错误提示应该就消失了。（如果仍然存在，请检查Java代码以及XML文件，确认是否存在输入或拼写错误。）

接下来，我们来编码使用按钮组件，这需要以下两个步骤：

- 引用生成的视图对象；
- 为对象设置监听器，以响应用户操作。

1.6.1　引用组件

在activity中，可调用以下Activity方法引用已生成的组件：

```
public View findViewById(int id)
```

该方法以组件的资源ID作为参数，返回一个视图对象。

在QuizActivity.java文件中，使用按钮的资源ID获取视图对象，赋值给对应的成员变量，如代码清单1-8所示。注意，赋值前，必须先将返回的View类型转换为Button。

代码清单1-8　引用组件（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

1.6.2　设置监听器

Android应用属于典型的事件驱动类型。不像命令行或脚本程序，事件驱动型应用启动后，即开始等待行为事件的发生，如用户点击某个按钮。（事件也可以由操作系统或其他应用触发，但用户触发的事件更直观，如点击按钮。）

应用等待某个特定事件的发生，也可以说应用正在“监听”特定事件。为响应某个事件而创建的对象叫作**监听器**（listener）。监听器会实现特定事件的**监听器接口**（listener interface）。

无需自己动手，Android SDK已经为各种事件内置了很多监听器接口。当前应用需要监听用户的按钮“点击”事件，因此监听器需实现View.OnClickListener接口。

首先处理TRUE按钮。在QuizActivity.java文件中，在onCreate(Bundle)方法的变量赋值语句后输入下列代码，如代码清单1-9所示。

代码清单1-9　为TRUE按钮设置监听器（QuizActivity.java）

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mTrueButton = (Button) findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Does nothing yet, but soon!
            }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

（如果遇到View cannot be resolved to a type的错误提示，请使用Option+Return（Alt+Enter）快捷键导入View类。）

在代码清单1-9中，我们设置了一个监听器。按钮mTrueButton被点击后，监听器会立即通知我们。传入setOnClickListener(OnClickListener)方法的参数是一个监听器。它是一个实现了OnClickListener接口的对象。

使用匿名内部类

这里，一个匿名内部类（anonymous inner class）实现了OnClickListener接口。语法看上去稍显复杂，不过有个助记小技巧：最外层一对括号内的全部代码就是传入setOnClickListener(OnClickListener)方法的参数。

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Does nothing yet, but soon!
    }
});
```

本书所有的监听器都以匿名内部类来实现。这样做有两大好处。第一，使用匿名内部类，可以相对集中地实现监听器方法，一眼可见；第二，事件监听器一般只在一个地方使用，使用匿名内部类，就不用去创建繁琐的命名类了。

匿名内部类实现了OnClickListener接口，因此它也必须实现该接口唯一的onClick(View)方法。onClick(View)现在是个空方法。虽然必须实现onClick(View)方法，但具体如何实现取决于使用者，因此即使是个空方法，编译器也可以编译通过。

（如果匿名内部类、监听器、接口等概念已忘得差不多了，现在就该去复习一下，或找本参考手册备查。）

参照代码清单1-10为FALSE按钮设置类似的事件监听器。

代码清单1-10 为FALSE按钮设置监听器（QuizActivity.java）

```
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Does nothing yet, but soon!
            }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Does nothing yet, but soon!
            }
        });
    }
```

1.7 创建提示消息

接下来要实现的是，分别点击两个按钮，弹出我们称之为toast的提示消息。Android的toast是用来通知用户的简短弹出消息，用户无需输入什么，也不用做任何干预操作。这里，我们要用toast来反馈答案，如图1-14所示。



图1-14 toast消息反馈

首先回到strings.xml文件，如代码清单1-11所示，为toast添加消息显示用的字符串资源。

代码清单1-11　增加toast字符串（strings.xml）

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>
</resources>
```

调用`Toast`类的以下方法，可创建toast：

```
public static Toast makeText(Context context, int resId, int duration)
```

该方法的`Context`参数通常是`Activity`的一个实例（`Activity`本身就是`Context`的子类）。第二个参数是toast要显示字符串消息的资源ID。`Toast`类必须借助`Context`才能找到并使用字符串资源ID。第三个参数通常是两个`Toast`常量中的一个，用来指定toast消息的停留时间。

创建toast后，可调用`Toast.show()`方法在屏幕上显示toast消息。

在`QuizActivity`代码里，分别调用`makeText(...)`方法，如代码清单1-12所示。在添加`makeText(...)`时，可利用Android Studio的代码自动补全功能，让代码输入更轻松。

使用代码自动补全

代码自动补全功能可以节约大量开发时间，越早掌握受益越多。

参照代码清单1-12，依次输入代码。当输入到`Toast`类后的点号时，Android Studio会弹出一个窗口，窗口内显示了建议使用的`Toast`类的常量与方法。

要选择需要的建议方法，使用上下键。（如果不想使用代码自动补全功能，请不要按Tab键、Return/Enter键，或使用鼠标点击弹出窗口，只管继续输入代码直至完成。）

在建议列表里，选择`makeText(Context context, int resID, int duration)`方法，代码自动补全功能会自动添加完整的方法调用。

完成`makeText`方法的全部参数设置，完成后的代码如代码清单1-12所示。

代码清单1-12　创建提示消息（QuizActivity.java）

```
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.correct_toast,
                       Toast.LENGTH_SHORT).show();
            // Does nothing yet, but soon!
    }
});
mFalseButton = (Button) find ViewById(R.id.false_button);
mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
```

```
            Toast.makeText(QuizActivity.this,
                           R.string.incorrect_toast,
                           Toast.LENGTH_SHORT).show();
            // Does nothing yet, but soon!
        }
    });
```

在makeText(...)里，传入QuizActivity实例作为Context的参数值。注意此处应输入的参数是QuizActivity.this，不要想当然地直接输入this。因为匿名类的使用，这里的this指的是监听器View.OnClickListener。

使用代码自动补全功能，自己也就不用导入Toast类了，因为Android Studio会自动导入相关类。

好了，现在可以运行应用了。

1.8　使用模拟器运行应用

运行Android应用需使用硬件设备或虚拟设备（virtual device）。包含在开发工具中的Android设备模拟器可提供多种虚拟设备。

要创建Android虚拟设备（AVD），在Android Studio中，选择Tools → Android → AVD Manager菜单项。AVD管理器窗口弹出时，点击窗口左下角的+Create Virtual Device...按钮。

在随后弹出的对话框中，可以看到有很多配置虚拟设备的选项。对于首个虚拟设备，我们选择模拟运行Nexus 5X设备，如图1-15所示。点击Next继续。

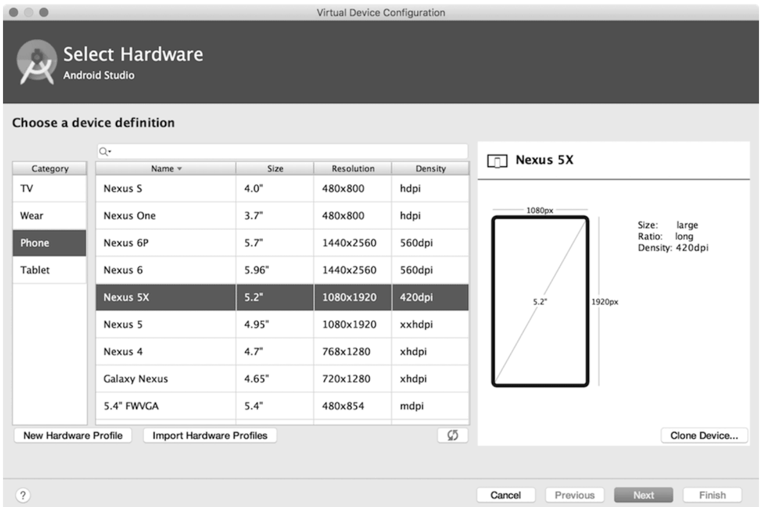


图1-15　创建新的AVD

如图1-16所示，接下来选择模拟器的系统镜像。选择x86 Nougat模拟器后点击Next按钮继续。

（在点击Next按钮之前，可能需要下载模拟器组件。按照提示操作即可。）

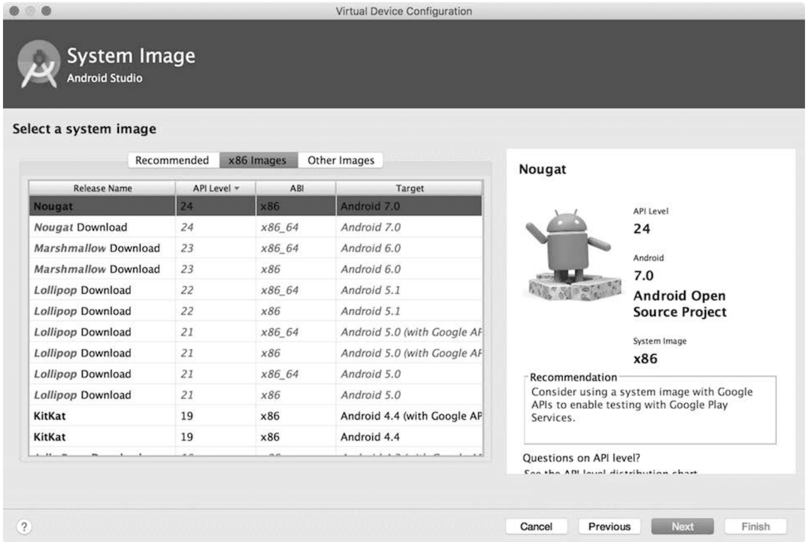


图1-16　选择系统镜像

最后，可以对模拟器的各项参数做最终修改并确认，如图1-17所示。当然，如果有需要，也可以事后再编辑修改模拟器的各项参数。现在，为模拟器取个便于识别的名称，然后点击Finish按钮完成虚拟设备的创建。



图1-17　模拟器参数调整

AVD创建成功后，我们用它运行GeoQuiz应用。点击Android Studio工具栏上的Run按钮，或者使用Control+R快捷键。Android Studio会自动找到新建的虚拟设备，安装应用包（APK），然后启动并运行应用。

模拟器的启动过程比较耗时，请耐心等待。等设备启动完成，应用运行后，就可以在应用界面点击按钮，让toast告诉你答案了。

假如启动时或在点击按钮时，GeoQuiz应用崩溃，可以在Android DDMS工具窗口的LogCat视图中看到有用的诊断信息。（如果LogCat没有自动打开，可点击Android Studio窗口底部的Android Monitor按钮打开它。）查看日志，可看到抢眼的红色异常信息，如图1-18所示。

```
Text
at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.NullPointerException
at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:21)
at android.app.Activity.performCreate(Activity.java:5008)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079)
```

图1-18　第21行代码处发生了`NullPointerException`异常

将输入的代码与书中的代码作一下比较，找出错误并修改，尝试重新运行应用（接下来的两章还会深入介绍LogCat和代码调试的知识）。

最好不要关掉模拟器，这样就不必在反复运行调试应用时，浪费时间等待AVD启动了。

单击AVD模拟器上的后退按钮可以停止应用。这个后退按钮的形状像一个指向左侧的三角形（在较早版本的Android中，它像一个U型箭头）。需要调试变更时，再通过Android Studio重新运行应用。

模拟器虽然好用，但在实体设备上测试应用能获得更准确的结果。在第2章中，我们会在实体设备上运行GeoQuiz应用，还会为GeoQuiz应用添加更多地理知识问题。

1.9　深入学习：Android 编译过程

学习到这里，你可能迫切想知道Android是如何编译的。你已经知道在项目文件发生变化时，Android Studio无需指示便会自动进行编译。在整个编译过程中，Android开发工具将资源文件、代码以及AndroidManifest.xml文件（包含应用的元数据）编译生成.apk文件。.apk文件要在模拟器上运行，还需以debug key签名。（分发.apk应用给用户时，应用必须以release key签名。更多有关编译过程的信息，可参考Android开发文档网页developer.android.com/tools/publishing/preparing.html。）

那么，应用的activity_quiz.xml布局文件的内容该如何转变为`View`对象呢？作为编译过程的一部分，aapt（Android Asset Packaging Tool）将布局文件资源编译压缩紧凑后，打包到.apk文件中。然后，在`QuizActivity`类的`onCreate(Bundle)`方法调用`setContentView(...)`方法时，`QuizActivity`使用`LayoutInflater`类实例化布局文件中定义的每一个`View`对象，如图1-19所示。

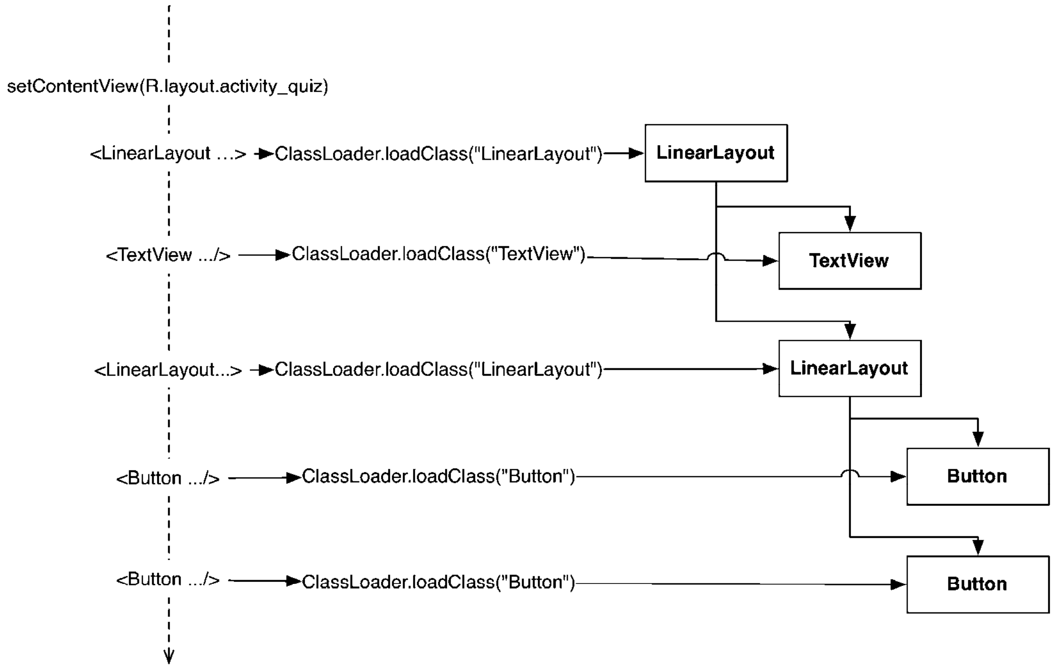


图1-19　activity_quiz.xml中的视图实例化

（除了在XML文件中定义视图外，也可以在activity里使用代码创建视图类。不过，从设计角度来看，应用展现层与逻辑层分离好处多多，其中最主要的一点是可以利用SDK内置的设备配置改变，这一点将在第3章中详细讲解。）

有关XML不同属性的工作原理以及视图如何显示在屏幕上等更多信息，请参见第9章。

Android 编译工具

当前，我们看到的项目编译都是在Android Studio里执行的。编译功能已整合到IDE中，IDE负责调用aapt等Android标准编译工具，但编译过程本身仍由Android Studio管理。

有时，出于某种原因，可能需要脱离Android Studio编译代码。最简单的方法是使用命令行编译工具。现代Android编译系统使用Gradle编译工具。

（注意，能读懂本小节内容并按步骤操作是最好的。如果看不懂，甚至不知道为什么要手工编译代码，或者是无法正确使用命令行，也不必太在意，请继续学习下一章内容。如何以及为什么要使用命令行工具，不在本书的讨论范围之内。）

要从命令行使用Gradle，请切换到项目目录并执行以下命令：

```
$ ./gradlew tasks
```

如果是Windows系统，执行以下命令：

```
> gradlew.bat tasks
```

执行以上命令会显示一系列可用任务。你需要的任务是installDebug，因此，再执行以下命令：

```
$ ./gradlew installDebug
```

如果是Windows系统，执行以下命令：

```
> gradlew.bat installDebug
```

以上命令将把应用安装到当前连接的设备上，但不会运行它。要运行应用，需要在设备上手动启动。

1.10　关于挑战练习

本书大部分章末都安排有挑战练习，需要你独立完成。有些很简单，就是练习所学知识。有些较难，需要较强的问题解决能力。

希望你一定完成这些练习。攻克它们不仅可以巩固所学知识，树立信心，还可以让自己从被动学习快速成长为自主开发的Android程序员。

尝试完成挑战练习时，若一时陷入困境，可休息休息，理理头绪，重新再来。如果仍然无法解决，可访问本书论坛 forums.bignerdranch.com，参考其他读者发布的解决方案。当然你也可以自己发布问题和答案并与其他读者一起交流学习。

为避免搞乱当前项目，建议你在Android Studio中先复制当前项目，然后在复制的项目上做练习。

在你的计算机上，通过文件浏览器找到项目文件的根目录，复制一份GeoQuiz文件并重命名为GeoQuiz Challenge。回到Android Studio中，选择File → Import Project....菜单项，通过导入功能找到GeoQuiz Challenge并导入。这样，复制项目就在新窗口中打开了。开始挑战吧！

1.11　挑战练习：定制 toast 消息

这个练习需要你定制toast消息，改在屏幕顶部而不是底部显示弹出消息。这需要使用`Toast`类的`setGravity`方法，并使用`Gravity.TOP`重力值。具体如何使用，请参考Android开发者文档。该方法所在网页为 developer.android.com/reference/android/widget/Toast.html#setGravity(int, int, int)。

第 2 章 Android与MVC设计模式

本章中，我们将升级GeoQuiz应用，提供更多的地理知识测试题目，如图2-1所示。

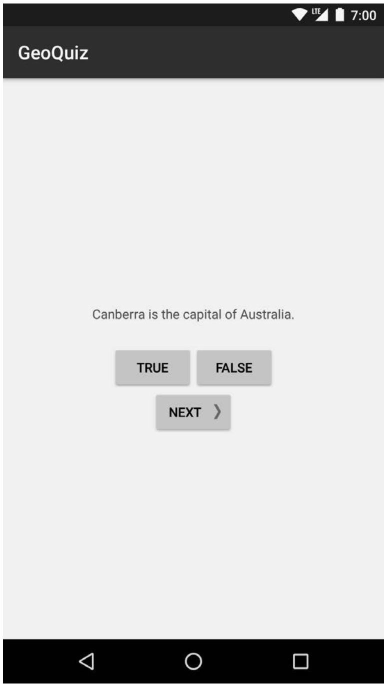


图2-1　更多测试题目

为实现目标，需要为GeoQuiz项目新增一个`Question`类。该类的一个实例代表一道题目。然后再创建一个`Question`数组对象交由`QuizActivity`管理。

2.1　创建新类

在项目工具窗口中，右键单击com.bignerdranch.android.geoquiz类包，选择New → Java Class菜单项。如图2-2所示，类名处填入`Question`，然后单击OK按钮。

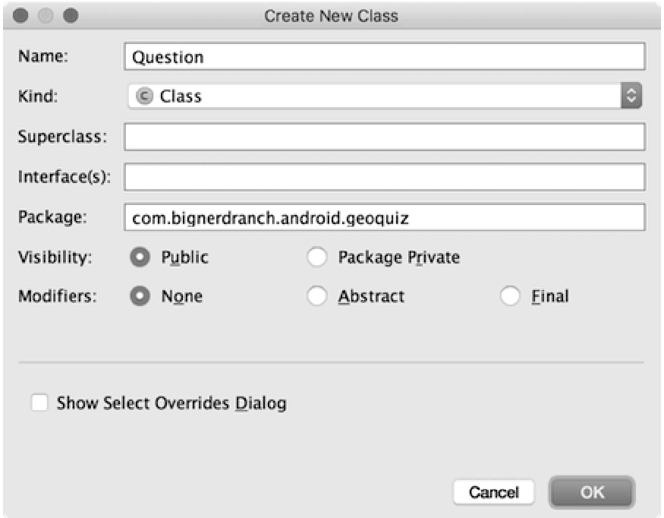


图2-2 创建Question类

在Question.java中，新增两个成员变量和一个构造方法，如代码清单2-1所示。

代码清单2-1 Question类中的新增代码（Question.java）

```
public class Question {

    private int mTextResId;
    private boolean mAnswerTrue;

    public Question(int textResId, boolean answerTrue) {
        mTextResId = textResId;
        mAnswerTrue = answerTrue;
    }
}
```

Question类中封装的数据有两部分：问题文本和问题答案（true或false）。

为什么mTextResId是int类型，而不是String类型呢？变量mTextResId用来保存地理知识问题字符串的资源ID。资源ID总是int类型，所以这里设置它为int类型。稍后会处理需要用到的问题字符串资源。

新增的两个变量需要getter方法与setter方法。为避免手工输入，可设置由Android Studio自动生成。

生成 getter 方法与 setter 方法

首先，配置Android Studio识别成员变量的m前缀。

打开Android Studio首选项对话框（Mac用户选择Android Studio菜单，Windows用户选择File → Settings菜单）。依次展开Editor和Code Style选项，在Java选项下选择Code Generation选项页。

在Naming表单的Field一行中，添加m作为前缀，如图2-3所示。然后添加s作为Static field的前缀。（GeoQuiz项目不会用到s前缀，但之后的项目会用到。）

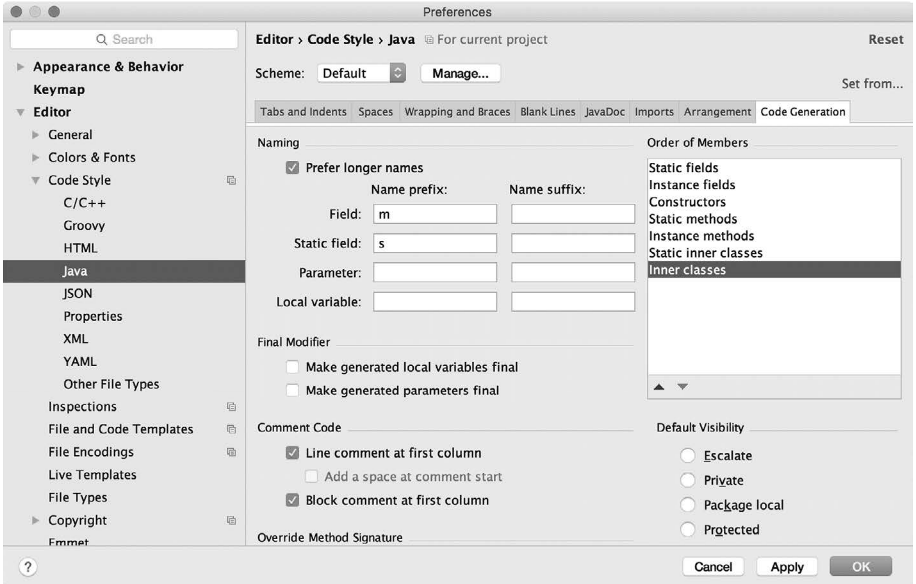


图2-3　设置Java代码风格首选项

单击OK按钮完成。

刚才设置的前缀有何作用？那就是，需要Android Studio为`mTextResId`生成获取方法时，它生成的是`getTextResId()`而不是`getMTextResId()`；而在为`mAnswerTrue`生成获取方法时，生成的是`isAnswerTrue()`而不是`isMAnswerTrue()`。

回到Question.java中，右击构造方法后方区域，选择Generate... → Getter and Setter菜单项。选择`mTextResId`和`mAnswerTrue`，为每个变量都生成getter方法与setter方法。单击OK按钮，Android Studio随即生成了getter与setter共4个方法的代码，如代码清单2-2所示。

代码清单2-2　生成getter方法与setter方法（Question.java）

```
public class Question {

    private int mTextResId;
    private boolean mAnswerTrue;
    ...
    public int getTextResId() {
        return mTextResId;
    }

    public void setTextResId(int textResId) {
        mTextResId = textResId;
    }
```

```
    public boolean isAnswerTrue() {
        return mAnswerTrue;
    }

    public void setAnswerTrue(boolean answerTrue) {
        mAnswerTrue = answerTrue;
    }

}
```

这样，Question类就完成了。稍后，我们会修改QuizActivity类来配合Question类使用。现在，先整体了解一下GeoQuiz应用，看看各个类是如何协同工作的。

我们使用QuizActivity创建Question数组对象，继而通过与TextView以及三个Button的交互，在屏幕上显示地理知识问题，并根据用户的回答作出反馈，如图2-4所示。

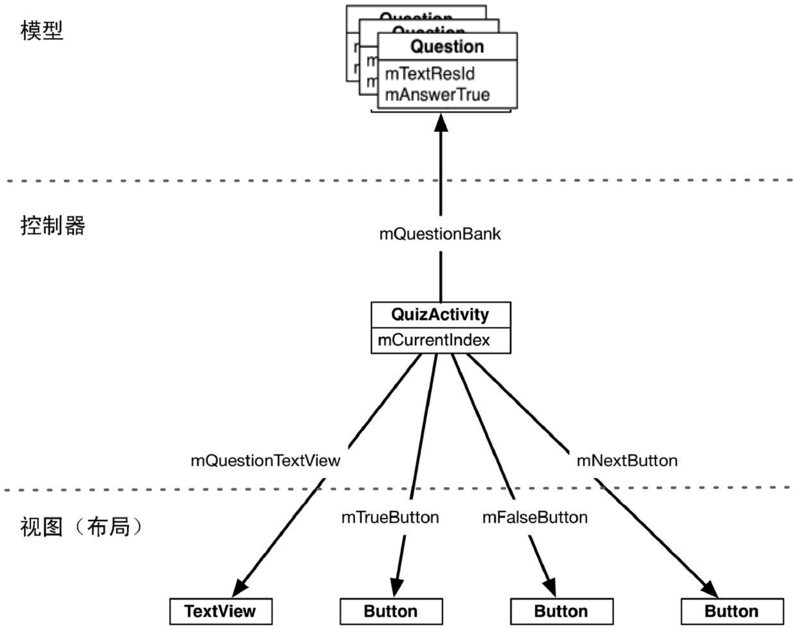


图2-4　GeoQuiz应用对象图解

2.2　Android 与 MVC 设计模式

如图2-4所示，应用对象分为模型、视图和控制器三类。Android应用基于模型–视图–控制器（Model-View-Controller，MVC）的架构模式进行设计。MVC设计模式表明，应用的任何对象，归根结底都属于模型对象、视图对象以及控制器对象中的一种。

- 模型对象存储着应用的数据和业务逻辑。模型类通常用来映射与应用相关的一些事物，如

用户、商店里的商品、服务器上的图片或者一段电视节目，抑或GeoQuiz应用里的地理知识问题。模型对象不关心用户界面，它为存储和管理应用数据而生。

Android应用里，模型类通常就是我们创建的定制类。应用的全部模型对象组成了*模型层*。GeoQuiz应用的模型层由`Question`类组成。

- 视图对象知道如何在屏幕上绘制自己，以及如何响应用户的输入，如触摸动作等。一个简单的经验法则是，凡是能够在屏幕上看见的对象，就是视图对象。

  Android自带很多可配置的视图类。当然，也可以定制开发其他视图类。应用的全部视图对象组成了*视图层*。

  GeoQuiz应用的视图层是由activity_quiz.xml文件中定义的各类组件构成的。

- 控制器对象含有应用的逻辑单元，是视图对象与模型对象的联系纽带。控制器对象响应视图对象触发的各类事件，此外还管理着模型对象与视图层间的数据流动。

  在Android的世界里，控制器通常是`Activity`、`Fragment`或`Service`的子类（第7章和第28章将分别介绍fragment和service的概念）。

  GeoQuiz应用的控制器层目前仅由`QuizActivity`类组成。

图2-5展示了在响应诸如单击按钮等用户事件时，对象间的交互控制数据流。注意，模型对象与视图对象不直接交互。控制器作为它们之间的联系纽带，接收对象发送的消息，然后向其他对象发送操作指令。

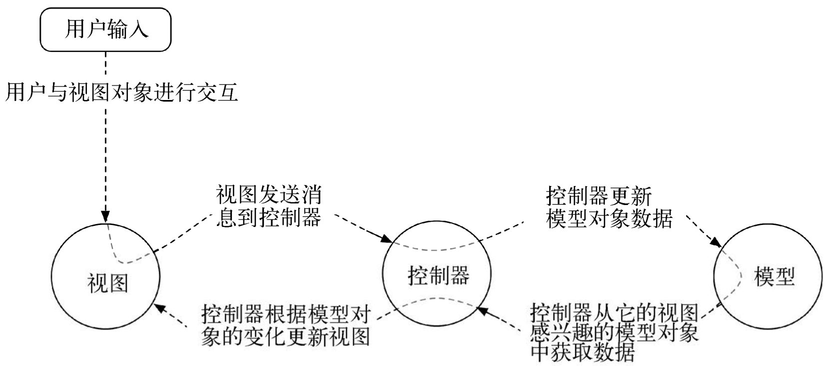


图2-5　MVC数据控制流与用户交互

使用 MVC 设计模式的好处

随着应用功能的持续扩展，应用往往会变得过于复杂而让人难以理解。以Java类组织代码有助于从整体视角设计和理解应用。这样，我们就可以按类而不是按变量和方法思考设计开发问题。

同样，把Java类以模型层、视图层和控制器层进行分类组织，也有助于我们设计和理解Android应用。这样，我们就可以按层而非一个个类来考虑设计开发了。

GeoQuiz应用虽不复杂，但以MVC分层模式设计它的好处还是显而易见的。接下来，我们会

升级GeoQuiz应用的视图层，为它添加一个NEXT按钮。你会发现，添加NEXT按钮时，可以不用考虑刚才创建的Question类。

MVC设计模式还便于复用类。相比功能多而全的类，功能单一的专用类更有利于代码复用。

举例来说，模型类Question与用作显示问题的组件毫无代码逻辑关联。这样，就很容易在应用里按需使用Question类。假设现在想显示包含所有地理知识问题的列表，很简单，直接利用Question对象逐条显示就可以了。

2.3　更新视图层

了解了MVC设计模式后，现在来更新GeoQuiz应用的视图层，为其添加一个NEXT按钮。

在Android的世界里，视图对象通常由XML布局文件生成。GeoQuiz应用唯一的布局定义在activity_quiz.xml文件中。布局定义文件需要更新的地方如图2-6所示。（注意，为节约版面，不变的组件属性就不再列出了。）

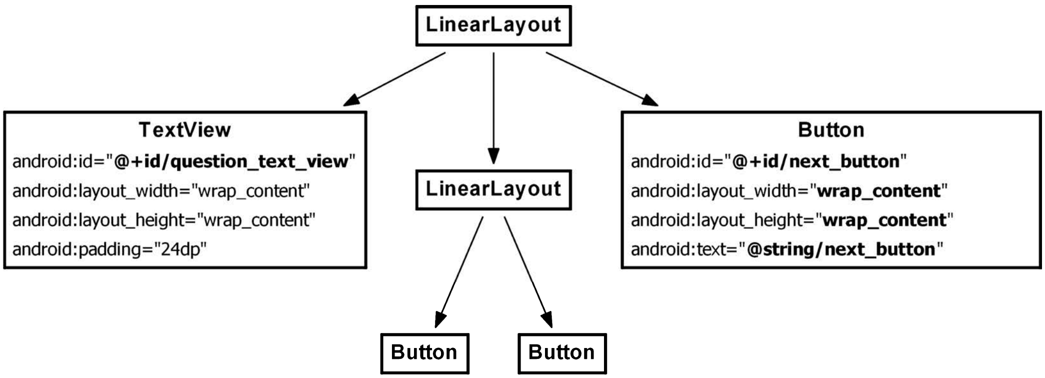


图2-6　新增按钮

应用视图层所需的改动如下。

- 删除TextView的android:text属性定义。再也不用硬编码地理知识问题了。
- 为TextView新增android:id属性。TextView组件需要资源ID，以便在QuizActivity代码中为它设置要显示的文字。
- 以根LinearLayout为父组件，新增一个Button组件。

回到activity_quiz.xml文件中，完成XML文件的相应修改，如代码清单2-3所示。

代码清单2-3　新增按钮以及对文本视图的调整（activity_quiz.xml）

```
<LinearLayout... >

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout ... >
        ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button" />

</LinearLayout>
```

保存activity_quiz.xml文件。这时，会看到一个错误提示，提醒缺少字符串资源。

回到res/values/strings.xml文件。重命名question_text，添加新按钮所需的字符串资源定义，如代码清单2-4所示。

代码清单2-4　更新字符串资源定义（strings.xml）

```
<string name="app_name">GeoQuiz</string>
<string name="question_text"> Canberra is the capital of Australia.</string>
<string name="question_australia">Canberra is the capital of Australia.</string>
<string name="true_button">True</string>
<string name="false_button">False</string>
<string name="next_button">Next</string>
<string name="correct_toast">Correct!</string>
```

既然已打开strings.xml文件，那就继续添加其他地理知识问题的字符串，结果如代码清单2-5所示。

代码清单2-5　新增问题字符串（strings.xml）

```
<string name="question_australia">Canberra is the capital of Australia.</string>
<string name="question_oceans">The Pacific Ocean is larger than
  the Atlantic Ocean.</string>
<string name="question_mideast">The Suez Canal connects the Red Sea
  and the Indian Ocean.</string>
<string name="question_africa">The source of the Nile River is in Egypt.</string>
<string name="question_americas">The Amazon River is the longest river
  in the Americas.</string>
<string name="question_asia">Lake Baikal is the world\'s oldest and deepest
  freshwater lake.</string>
...
```

注意最后一个字符串定义中的\'。为得到符号'，这里使用了转义字符。在字符串资源定义中，也可使用其他常见的转义字符，比如\n是指新行符。

保存修改过的文件，然后回到activity_quiz.xml文件中，在图形布局工具里预览修改后的布局文件。

至此，GeoQuiz应用视图层的更新就全部完成了。为让GeoQuiz应用跑起来，接下来要更新

控制层的QuizActivity类。

2.4 更新控制器层

在上一章，应用控制器层的QuizActivity类的处理逻辑很简单：显示定义在activity_quiz.xml文件中的布局对象，为两个按钮设置监听器，实现响应用户点击事件并创建toast消息。

既然现在有更多的地理知识问题可以检索与展示，QuizActivity类就需要更多的处理逻辑来让GeoQuiz应用的模型层与视图层协作。

打开QuizActivity.java文件，添加TextView和新Button变量。另外，再创建一个Question对象数组以及一个该数组的索引变量，如代码清单2-6所示。

代码清单2-6 增加按钮变量及Question对象数组（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;
    private Button mNextButton;
    private TextView mQuestionTextView;

    private Question[] mQuestionBank = new Question[] {
        new Question(R.string.question_australia, true),
        new Question(R.string.question_oceans, true),
        new Question(R.string.question_mideast, false),
        new Question(R.string.question_africa, false),
        new Question(R.string.question_americas, true),
        new Question(R.string.question_asia, true)
    };

    private int mCurrentIndex = 0;
    ...
```

这里，我们通过多次调用Question类的构造方法，创建了Question对象数组。

（在更为复杂的项目里，这类数组的创建和存储会单独处理。在后续应用开发中，会介绍更好的模型数据存储管理方式。现在，简单起见，我们选择在控制器层代码中创建数组。）

要在屏幕上显示一系列地理知识问题，可以使用mQuestionBank数组、mCurrentIndex变量以及Question对象的存取方法。

首先，引用TextView，并将其文本内容设置为当前数组索引所指向的地理知识问题，如代码清单2-7所示。

代码清单2-7 使用TextView（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_quiz);

        mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);

        mTrueButton = (Button) findViewById(R.id.true_button);
        ...
    }
}
```

保存所有文件，确保没有错误发生。然后运行GeoQuiz应用。可看到数组存储的第一个问题显示在TextView上了。

现在我们来处理NEXT按钮。首先引用NEXT按钮，然后为其设置监听器View.OnClickListener。该监听器的作用是：递增数组索引并相应地更新TextView的文本内容，如代码清单2-8所示。

代码清单2-8　使用新增的按钮（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            ...
            }
        });

        mNextButton = (Button) findViewById(R.id.next_button);
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mCurrentIndex].getTextResId();
                mQuestionTextView.setText(question);
            }
        });
    }
}
```

可以看到，更新mQuestionTextView变量的相同代码分布在两个不同的地方。参照代码清单2-9,花点时间把公共代码放在单独的私有方法里,然后在mNextButton监听器里以及onCreate(Bundle)方法的末尾分别调用它，以初步设置activity视图中的文本。

代码清单2-9　使用updateQuestion()封装公共代码（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        ...
        mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
        ...
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mCurrentIndex].getTextResId();
                mQuestionTextView.setText(question);
                updateQuestion();
            }
        });

        updateQuestion();
    }

    private void updateQuestion() {
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }
}
```

运行GeoQuiz应用，验证新增的NEXT按钮。

一切正常的话，问题应该已经完美显示出来了。当前，GeoQuiz应用认为所有问题的答案都是true，下面着手修正这个逻辑错误。同样，为避免代码重复，我们将解决方案封装在一个私有方法里。

要添加到QuizActivity类的方法如下：

```
private void checkAnswer(boolean userPressedTrue)
```

该方法接受布尔类型的变量参数，判别用户单击了TRUE还是FALSE按钮。然后，将用户的答案同当前Question对象中的答案作比较。最后，判断答案正确与否，生成一个toast消息反馈给用户。

在QuizActivity.java文件中，添加checkAnswer(boolean)方法的实现代码，如代码清单2-10所示。

代码清单2-10　增加checkAnswer(boolean)方法（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    private void updateQuestion() {
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }
```

```
    private void checkAnswer(boolean userPressedTrue) {
        boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();

        int messageResId = 0;

        if (userPressedTrue == answerIsTrue) {
            messageResId = R.string.correct_toast;
        } else {
            messageResId = R.string.incorrect_toast;
        }

        Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
            .show();
    }
}
```

在按钮的监听器里，调用checkAnswer(boolean)方法，如代码清单2-11所示。

代码清单2-11　调用checkAnswer(boolean)方法（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mTrueButton = (Button) findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                               R.string.correct_toast,
                               Toast.LENGTH_SHORT).show();
                checkAnswer(true);
            }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                               R.string.incorrect_toast,
                               Toast.LENGTH_SHORT).show();
                checkAnswer(false);
            }
        });
        ...
    }
    ...
}
```

GeoQuiz应用可以运行了，这次，我们要在物理设备上运行它。

2.5　在物理设备上运行应用

本节，我们学习如何设置系统、设备和应用，实现在硬件设备上运行GeoQuiz应用。

2.5.1　连接设备

首先，将设备连接到系统上。Mac系统应该会立即识别出所用设备。如果是Windows系统，则可能需要安装adb（Android Debug Bridge）驱动。如果Windows系统自身无法找到adb驱动，请到设备生产商的网站下载。

2.5.2　配置设备用于应用开发

要在设备上测试应用，首先应打开设备的USB调试模式。

开发者选项默认不可见。先选择“设定→关于平板/手机”项，再点击版本号（Build Number）7次启用它，然后回到“设定”项，选择“开发”项，找到并勾选“USB调试”选项。

不同版本设备的设置方法迥异。如果在设置过程中遇到问题，请访问以下网页求助：http://developer.android.com/tools/device.html。

可打开Devices视图确认设备是否已识别。选择Android Studio底部的Android Monitor工具窗口，可快速打开Devices视图。设备连接成功的话，你会看到已连接设备的下拉列表。AVD以及硬件设备应该就列在其中，如图2-7所示。

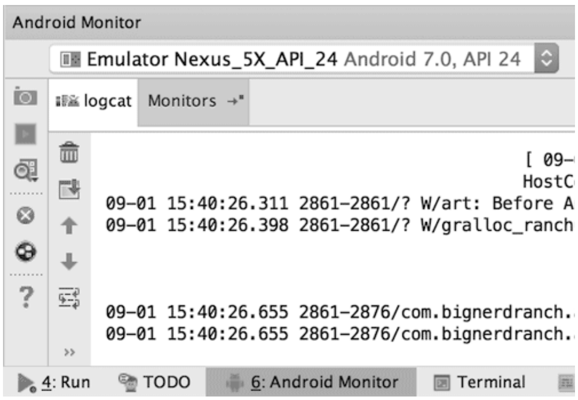


图2-7　查看已连接设备

如果设备无法识别，请首先确认是否已打开设备和开发者选项。如果仍然无法解决，请访问Android开发网站http://developer.android.com/tools/device.html，或访问本书论坛http://forums.bignerdranch.com求助。

再次运行GeoQuiz应用，Android Studio会询问是在虚拟设备还是物理设备上运行应用。选择物理设备并继续。稍等片刻，GeoQuiz应用应该已经在设备上运行了。

如果Android Studio没有给出选项，应用依然在虚拟设备上运行，请按以上步骤重新检查设备设置，并确保设备与系统已正确连接。然后，再检查运行配置是否有问题。要修改运行配置，请选择Android Studio窗口靠近顶部的app下拉列表，如图2-8所示。

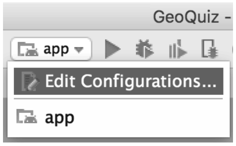


图2-8 打开运行配置

选择Edit Configurations…打开运行配置编辑窗口，如图2-9所示。

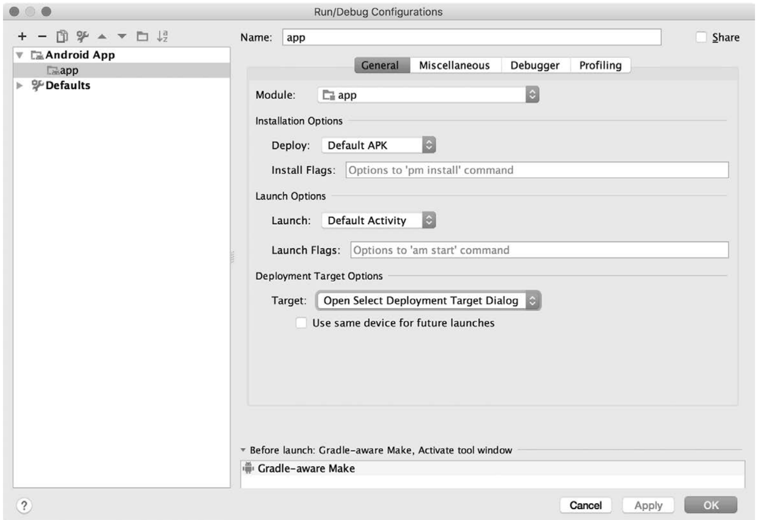


图2-9 运行配置界面

选择窗口左边区域的app，确认已选中Deployment Target Options区域的Open Select Deployment Target Dialog选项。点击OK按钮并重新运行应用，现在应能看到可以运行应用的设备选项了。

2.6 添加图标资源

GeoQuiz应用现在已经可用了。如果NEXT按钮上能够显示向右的图标，用户界面看起来应该会更美。

本书随书文件中提供了这样的箭头图标（https://www.bignerdranch.com/solutions/AndroidProgramming3e.zip）。随书文件集合了Android Studio项目文件，每章对应一个项目文件。

下载随书文件，找到并打开02_MVC/GeoQuiz/app/src/main/res目录。在该目录下，可以看到

drawable-mdpi、drawable-hdpi、drawable-xhdpi和drawable-xxhdpi四个目录。

四个目录各自的后缀名代表设备的像素密度。

- mdpi：中等像素密度屏幕（约160dpi）。
- hdpi：高像素密度屏幕（约240dpi）。
- xhdpi：超高像素密度屏幕（约320dpi）。
- xxhdpi：超超高像素密度屏幕（约480dpi）。

（另外还有ldpi和xxxhdpi这两个本书用不到的类别，因此，未包括在内。）

每个目录下，有两个图片文件：arrow_right.png和arrow_left.png。这些图片文件都是按照目录名对应的dpi定制的。

GeoQuiz项目中的所有图片资源都会随应用安装在设备里，Android操作系统知道如何为不同设备提供最佳匹配。注意，在为不同设备准备适配图片的同时，应用安装包容量也随之增大。当然，对于GeoQuiz这样的小项目，问题并不明显。

如果应用不包含设备对应的屏幕像素密度文件，在运行时，Android系统会自动找到可用的图片资源，针对该设备进行缩放适配。有了这种特性，就不一定要准备各种屏幕像素密度文件了。因此，为控制应用包的大小，可以只为主流设备准备分辨率较高的定制图片资源。至于那些不常见的低分辨率设备，让Android系统自动适配就好。

（第23章会介绍为屏幕像素密度定制图片的替代方案，另外，还会解释mipmap目录的用途。）

2.6.1 向项目中添加资源

接下来，需将图片文件添加到GeoQuiz项目资源中去。

首先，确认打开了Android Studio的Project视图（打开方式请参见图1-13）。展开GeoQuiz/app/src/main/res目录，会看到名为mipmap-hdpi和mipmap-xhdpi这样的目录，如图2-10所示。

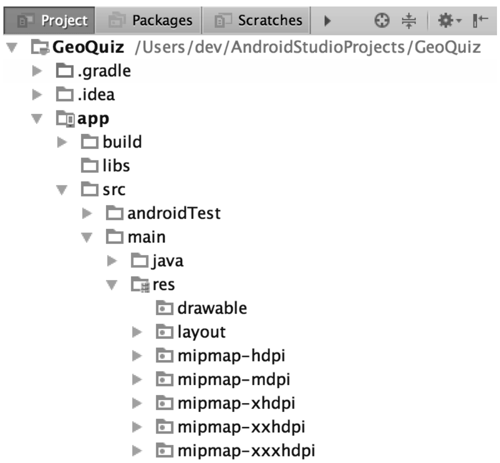


图2-10 还缺各类drawable目录

在随书文件中，选择并复制drawable-mdpi、drawable-hdpi、drawable-xhdpi和drawable-xxhdpi这四个目录。然后粘贴到Android Studio的app/src/main/res目录中。完成后，在Android Studio的项目工具窗口，就可以看到这四个目录，每个目录当中含有对应的arrow_left.png和arrow_right.png文件，如图2-11所示。

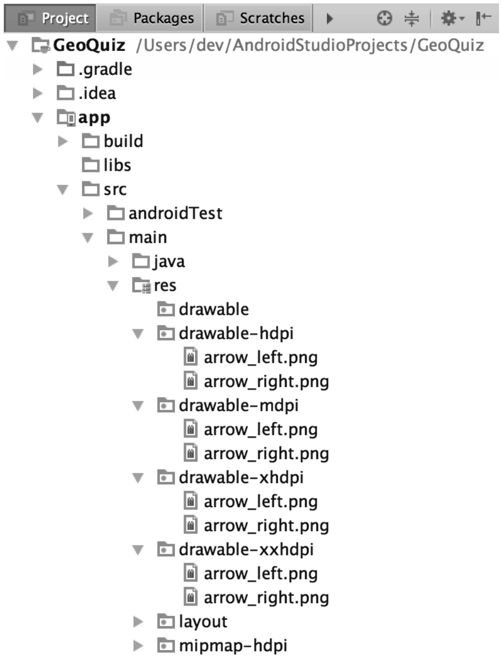


图2-11　drawable目录中的箭头图标文件

如果将项目工具窗口切换回Android视图，新增加的drawable图片资源会以图2-12所示的形式展示。

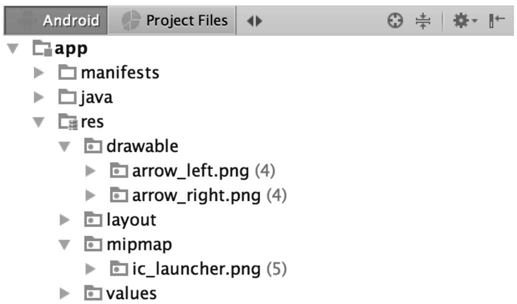


图2-12　drawable目录中的箭头图标文件汇总

向应用添加图片就这么简单。任何添加到res/drawable目录中、后缀名为.png、.jpg或者.gif的文件都会自动获得资源ID。（注意，文件名必须是小写字母且不能有任何空格符号。）

这些资源ID并不按照屏幕像素密度匹配，因此不需要在运行的时候确定设备的屏幕像素密度，只需在代码中引用这些资源ID就可以了。应用运行时，操作系统知道如何在特定的设备上显示匹配的图片。

Android资源系统是如何工作的？从第3章起，我们会深入学习这方面的相关知识。现在，能显示右箭头图标就可以了。

2.6.2 在 XML 文件中引用资源

在代码中，可以使用资源ID引用资源。如果想在布局定义中配置NEXT按钮显示箭头图标，又该如何在布局XML文件中引用资源呢？

很简单，只是语法稍有不同而已。打开activity_quiz.xml文件，为`Button`组件新增两个属性，如代码清单2-12所示。

代码清单2-12　为NEXT按钮增加图标（activity_quiz.xml）

```
<LinearLayout  ... >
    ...
    <LinearLayout  ... >
        ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp" />

</LinearLayout>
```

在XML资源文件中，通过资源类型和资源名称，可引用其他资源。以`@string/`开头的定义是引用字符串资源。以`@drawable/`开头的定义是引用drawable资源。

自第3章起，我们还会学习更多资源命名以及res目录结构中其他资源的使用等相关知识。

运行GeoQuiz应用。新按钮很漂亮吧？测试一下，确认它仍然工作正常。

别高兴得太早，GeoQuiz应用有个bug。应用运行时，单击NEXT按钮显示下一道题，然后旋转设备。如果是模拟器，请点击浮动工具栏上的左旋或右旋按钮，如图2-13所示。

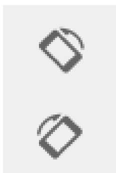


图2-13　控制旋转

发现没有，设备旋转后，应用又显示了第一道测试题。怎么回事？如何修正？

要解决此类问题，需了解activity生命周期的概念。第3章会专门介绍。

2.7　挑战练习：为 TextView 添加监听器

NEXT按钮不错，但如果用户单击应用的TextView文字区域（地理知识问题），也可以跳转到下一道题，用户体验会更好。

提示　TextView也是View的子类，因此和Button一样，可为TextView设置View.OnClick-Listener监听器。

2.8　挑战练习：添加后退按钮

为GeoQuiz应用新增后退按钮（PREV），用户单击时，可以显示上一道测试题目。完成后的用户界面应如图2-14所示。

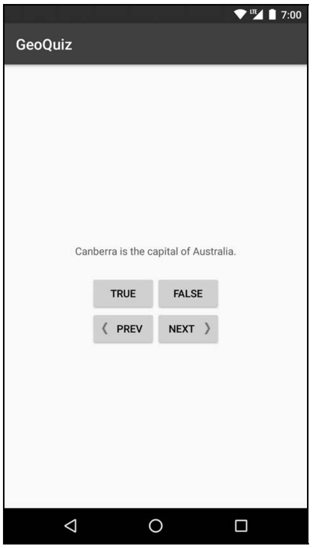


图2-14　添加了后退按钮的用户界面

这是个很棒的练习，需回顾本章和上一章的内容才能完成。

2.9　挑战练习：从按钮到图标按钮

如果前进与后退按钮上只显示指示图标，用户界面更清爽，如图2-15所示。



图2-15 只显示图标的按钮

要完成此练习，需将普通的Button组件替换成ImageButton组件。

ImageButton组件继承自ImageView。Button组件则继承自TextView。ImageButton和Button与View间的继承关系如图2-16所示。

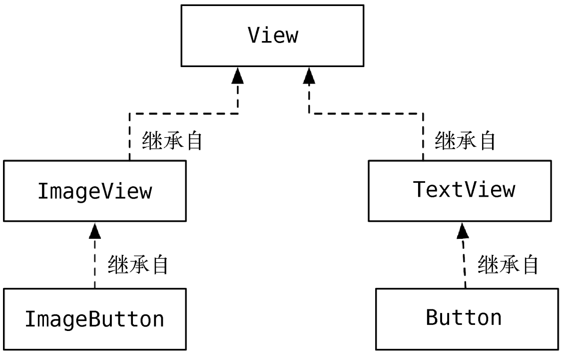


图2-16 ImageButton和Button与View间的继承关系

如以下代码所示，以ImageButton组件替换Button组件，删除NEXT按钮的text以及drawable属性定义，并添加ImageView属性：

```
<Button ImageButton
  android:id="@+id/next_button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/next_button"
```

```
android:drawableRight="@drawable/arrow_right"
android:drawablePadding="4dp"
android:src="@drawable/arrow_right"
/>
```

当然，为了使用ImageButton，还要调整QuizActivity类的代码。

将Button组件替换成ImageButton后，Android Studio会警告说找不到android:content-Description属性定义。该属性能为视力障碍用户提供方便。在为其设置文字属性值后，如果设备的可访问性选项作了相应设置，那么在用户点击图形按钮时，设备便会读出属性值的内容。

最后，为每个ImageButton都添加上android:contentDescription属性定义。

第 3 章 activity的生命周期

你在第2章章末已看到，只要一旋转设备，地理知识问题就回到第一道初始题。用户旋转设备，应用状态就重置，是什么原因呢？要想搞明白，并解决这类常见的旋转问题，首先要学习activity生命周期的基础知识。

每个`Activity`实例都有其生命周期。在其生命周期内，activity在运行、暂停、停止和不存在这四种状态间转换。每次状态转换时，都有相应的`Activity`方法发消息通知activity。图3-1显示了activity的生命周期、状态以及状态切换时系统调用的方法。

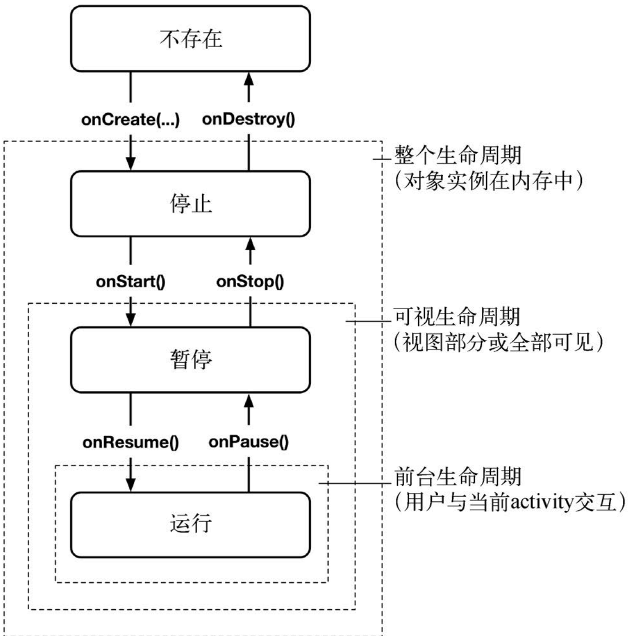


图3-1　activity的状态图解

内存中有没有activity的实例，用户是否看得到，是否活跃在前台（等待或接受用户输入中），看图3-1的各种状态就知道了。完整总结如表3-1所示。

表3-1　activity的状态

状　　态	有内存实例	用户可见	处于前台
不存在	否	否	否
停止	是	否	否
暂停	是	是或部分*	否
运行	是	是	是

* 某些场景下，暂停状态的activity可能会部分或完全可见，详见3.1.3节。

用户可以与当前运行状态下的activity交互。设备上有很多应用，但是，任何时候只能有一个activity处于用户能交互的运行状态。

借助图3-1所示的方法，Activity的子类可以在activity的生命周期状态发生关键性转换时完成某些工作。这些方法通常被称为生命周期回调方法。

我们已熟悉这些方法中的onCreate(Bundle)方法。在创建activity实例后，但在此实例出现在屏幕上之前，Android操作系统会调用该方法。

通常，通过覆盖onCreate(Bundle)方法，activity可以预处理以下UI相关工作：

- 实例化组件并将它们放置在屏幕上（调用setContentView(int)方法）；
- 引用已实例化的组件；
- 为组件设置监听器以处理用户交互；
- 访问外部模型数据。

切记，千万不要自己去调用onCreate(Bundle)方法或任何其他activity生命周期方法。为通知activity状态变化，你只需在Activity子类里覆盖这些方法，Android会适时调用它们（看当前用户状态以及系统运行情况）。

3.1　日志跟踪理解 activity 生命周期

本节，我们会覆盖一些activity生命周期方法，以此一窥究竟，学习并理解QuizActivity的生命周期。这些覆盖方法会输出日志，告诉我们操作系统何时调用了它们。这样，伴随用户操作，QuizActivity的状态如何变化，就很清楚了。

3.1.1　输出日志信息

Android的android.util.Log类能够向系统级共享日志中心发送日志信息。Log类有好几个日志记录方法。本书用得最多的是以下方法：

```
public static int d(String tag, String msg)
```

d代表“debug”，用来表示日志信息的级别。（本章最后一节会详细讲解有关Log级别的内容。）第一个参数是日志的来源，第二个参数是日志的具体内容。

该方法的第一个参数通常以类名为值的TAG常量传入。这样，就很容易看出日志信息的来源。在QuizActivity.java中，为QuizActivity类新增一个TAG常量，如代码清单3-1所示。

代码清单3-1　新增一个TAG常量（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private static final String TAG = "QuizActivity";
    ...
}
```

然后，在onCreate(Bundle)方法里调用Log.d(...)方法记录日志，如代码清单3-2所示。

代码清单3-2　为onCreate(Bundle)方法添加日志输出代码（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);
        ...
    }
}
```

接下来，在QuizActivity类的onCreate(Bundle)之后，覆盖其他五个生命周期方法，如代码清单3-3所示。

代码清单3-3　覆盖更多生命周期方法（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedinstanceState) {
        ...
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart() called");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume() called");
    }

    @Override
    public void onPause() {
        super.onPause();
```

```
        Log.d(TAG, "onPause() called");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop() called");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy() called");
    }
    ...
}
```

注意，我们先是调用了超类的实现方法，然后才调用具体的日志记录方法。这些超类方法的调用不可或缺。从以上代码可以看出，在回调覆盖实现方法里，超类实现方法总在第一行调用。也就是说，应首先调用超类实现方法，然后再调用其他方法。

知道为什么要使用@Override注解吗？使用@Override注解，就是要求编译器保证当前类拥有你要覆盖的方法。例如，对于如下拼写错误的方法，编译器会发出警告：

```
public class QuizActivity extends AppCompatActivity {

    @Override
    public void onCreat(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
    }
  ...
}
```

AppCompatActivity父类没有onCreat(Bundle)方法，因此编译器发出了警告。这样，你就能及时改正拼写错误，而不是等到应用运行时，才发现奇怪的现象，再去查找问题在哪。

3.1.2　使用 LogCat

应用运行时，可以使用LogCat工具查看日志。LogCat是Android SDK工具中的一款日志查看器。

运行GeoQuiz应用时，应该能在Android Studio底部看见LogCat，如图3-2所示。如果看不到，请切换至Android Monitor工具窗口模式，并确保已选中logcat选项页。

运行GeoQuiz应用，立刻可看到LogCat窗口中的各类混杂信息。这些日志中，有些是以应用包名为默认名的应用类信息，有些是系统输出信息。

为方便查找，可使用TAG常量过滤日志输出。单击LogCat面板右上角写着Show only selected application的下拉列表。这是过滤项下拉列表。当前选项控制只显示来自应用的日志信息。如果选No Filters，则会看到系统的所有输出信息。

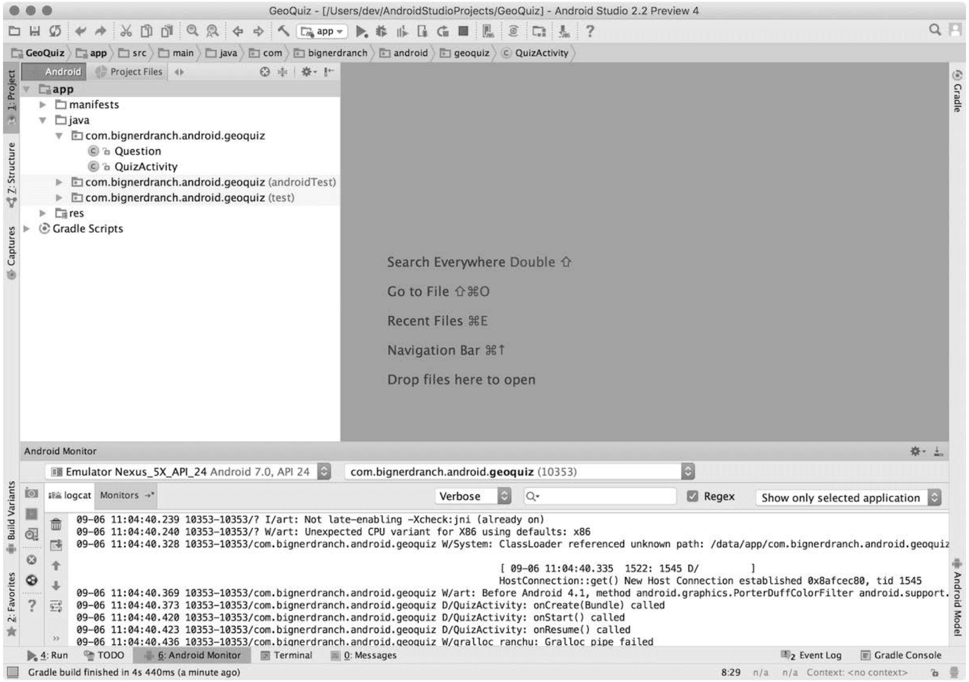


图3-2 Android Studio中的LogCat

要创建过滤设置，选择Edit Filter Configuration选项。单击绿色+按钮，创建一个消息过滤器。在Filter Name处输入`QuizActivity`，Log Tag处同样输入`QuizActivity`，如图3-3所示。

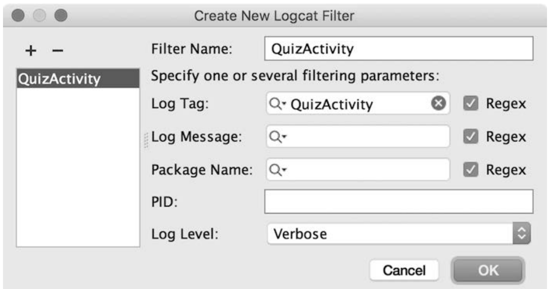


图3-3 在LogCat中创建过滤器

单击OK按钮。现在，LogCat窗口仅显示Tag为`QuizActivity`的日志信息，如图3-4所示。

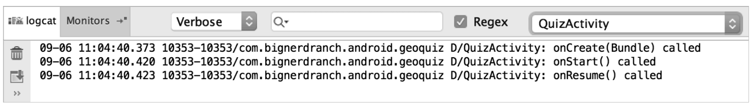


图3-4 应用启动后，被调用的三个生命周期方法

3.1.3　activity 生命周期实例解析

如图3-4所示，GeoQuiz应用启动并创建QuizActivity初始实例后，onCreate(Bundle)、onStart()和onResume()这三个生命周期方法被调用了。QuizActivity实例现在处于运行状态（在内存里，用户可见，活动在前台）。

（如果看不到过滤后的信息列表，请选择LogCat过滤器下拉列表中的QuizActivity过滤项。）

下面我们来做个有趣的实验。在设备上单击后退键，再查看LogCat。可以看到，日志显示QuizActivity的onPause()、onStop()和onDestroy()方法被调用了，如图3-5所示。QuizActivity实例处于不存在的状态（不在内存里，显然不可见，自然不会活动在前台）。

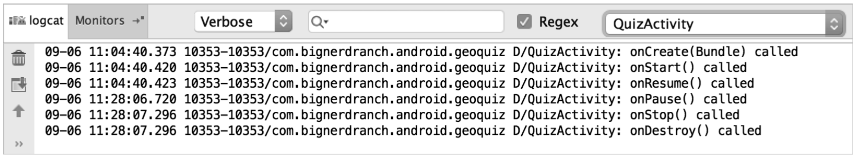


图3-5　单击后退键销毁activity

单击设备的后退键，相当于告诉Android系统："activity已用完，现在不需要它了。"随即，系统就销毁了该activity的视图及其内存里的相关信息。这实际是Android系统节约使用设备有限资源的一种方式。

点击GeoQuiz应用图标，再次运行它。日志显示，Android创建了全新的QuizActivity实例，然后调用onCreate()、onStart()和onResume()方法。QuizActivity从不存在变为运行状态。

现在，单击主屏幕键，随即，主屏界面出现了，QuizActivity视图不见了。QuizActivity处于啥状态呢？查看LogCat，可以看到系统调用了QuizActivity的onPause()和onStop()方法，但并没有调用onDestroy()方法，如图3-6所示。

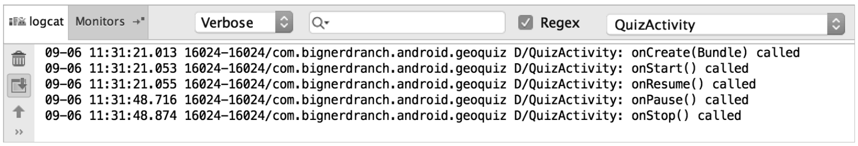


图3-6　单击主屏幕键停止activity

单击主屏幕键，相当于告诉Android系统："我去别处看看，稍后可能回来。"此时，Android系统会先暂停，再停止当前activity。这表明，QuizActivity实例已处于停止状态（在内存中，但不可见，不会活动在前台）。这样，Android系统就能快速响应，重启QuizActivity，回到用户离开时的状态。

（需要注意的是，停止的activity能够存在多久，谁也无法保证。系统需要回收内存时，它将首先销毁那些停止的activity。）

现在，我们调出设备的概览屏。如果是比较新的设备，可单击主屏幕键旁的最近应用键，调

出概览屏，如图3-7所示。（如果设备没有最近应用键，则长按主屏幕键调出概览屏。）



图3-7 主屏幕键、后退键以及最近应用键

概览屏的每张卡片代表用户之前交互过的一个应用，如图3-8所示。（用户常把概览屏称作最近应用屏或任务管理器。不过，既然Google开发者文档将其称作概览屏，本书从之。）

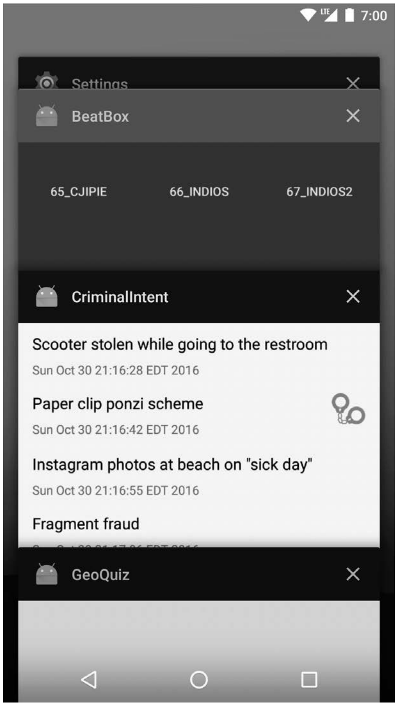


图3-8 概览屏

在概览屏中，单击GeoQuiz应用，`QuizActivity`视图随即出现。

LogCat日志显示，系统没有调用`onCreate()`方法（因为`Activity`实例还在内存里，自然不

用重建了），而是调用了onStart()和onResume()方法。用户按了主屏幕键后，QuizActivity最后进入停止状态，再次调出应用时，QuizActivity只需要重新启动（进入暂停状态，用户可见），然后继续运行（进入运行状态，活动在前台）。

activity有时也会一直处于暂停状态（用户完全或部分可见，但不在前台）。可能出现部分可见暂停状态的场景：在一个activity之上启动带透明背景视图或小于屏幕尺寸视图的新activity时。可能出现完全可见暂停状态的场景：应用多窗口模式下（Android 6.0及更高系统版本才支持），当前activity在一个窗口完全可见，而用户在不包含当前activity的另一个窗口操作时。

在本书的后续学习过程中，为完成各种现实任务，我们还会覆盖一些其他生命周期方法，进一步学习更多生命周期方法的用法。

3.2　设备旋转与 activity 生命周期

现在，可以处理第2章结束时发现的应用缺陷了。启动GeoQuiz应用，单击NEXT按钮显示第二道地理知识问题，然后旋转设备。（模拟器的旋转，使用Command+右方向键/Ctrl+右方向键，或点击工具栏上的旋转按钮。）

设备旋转后，GeoQuiz应用又回到了第一道问题。查看LogCat日志看看发生了什么，如图3-9所示。



图3-9　QuizActivity已死，QuizActivity万岁

设备旋转时，系统会销毁当前QuizActivity实例，然后创建一个新的QuizActivity实例。再次旋转设备，又一次见证这个销毁与再创建的过程。

这就是问题所在。每次旋转设备，当前QuizActivity实例会完全销毁，实例中的mCurrentIndex当前值会从内存里被抹掉。旋转后，Android重新创建了QuizActivity新实例，mCurrentIndex在onCreate(Bundle)方法中被初始化为0。一切重头再来，用户又看到第一道题。

稍后会修正这个缺陷。不要停留在问题表面，接下来，由表及里，深入分析为什么有此问题。

设备配置与备选资源

旋转设备会改变设备配置（device configuration）。设备配置实际是一系列特征组合，用来描述设备当前状态。这些特征有：屏幕方向、屏幕像素密度、屏幕尺寸、键盘类型、底座模式以及语言等。

通常，为匹配不同的设备配置，应用会提供不同的备选资源。为适应不同分辨率的屏幕，向

项目添加多套箭头图标就是这样一个使用案例。

设备的屏幕像素密度是个固定的设备配置，无法在运行时发生改变。然而，屏幕方向等特征可以在应用运行时改变。

在运行时配置变更（runtime configuration change）发生时，可能会有更合适的资源来匹配新的设备配置。于是，Android销毁当前activity，为新配置寻找最佳资源，然后创建新实例使用这些资源。眼见为实，下面为设备配置变更新建备选资源，只要设备旋转至水平方位，Android就会自动发现并使用它。

创建水平模式布局

在项目工具窗口中，右键单击res目录后选择New→Android resource directory菜单项。创建资源目录界面列出了资源类型及其对应的资源特征，如图3-10所示。从资源类型（Resource type）列表中选择layout，保持Source set的main选项不变。

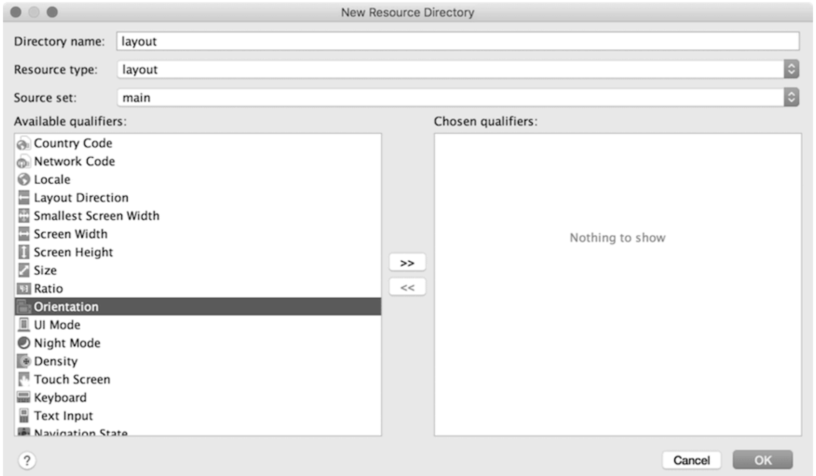


图3-10　创建新的资源目录

接下来选中待选资源特征列表中的Orientation，然后单击>>按钮将其移动至已选资源特征（Chosen qualifiers）区域。

最后，确认选中Screen orientation下拉列表中的Landscape选项，并确保目录名（Directory name）显示为layout-land，如图3-11所示。这个窗口看起来有模有样，但实际用途仅限于设置目录名。点击OK按钮让Android Studio创建res/layout-land。

这里的-land后缀名是配置修饰符的另一个使用例子。Android依靠res子目录的配置修饰符定位最佳资源以匹配当前设备配置。访问Android开发网页http://developer.android.com/guide/topics/resources/providing-resources.html，可查看Android的配置修饰符列表及其代表的设备配置信息。

设备处于水平方向时，Android会找到并使用res/layout-land目录下的布局资源。其他情况下，它会默认使用res/layout目录下的布局资源。然而，目前在res/layout-land目录下并没有布局资源。让我们解决这个问题。

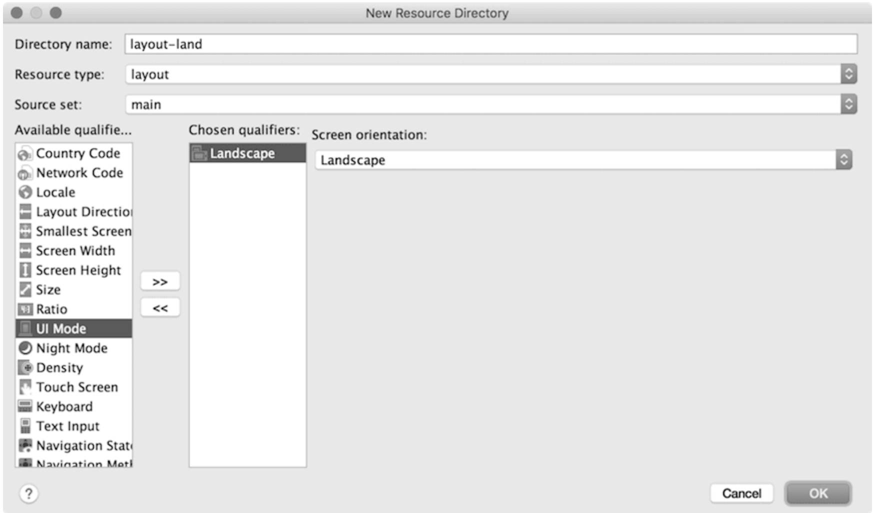


图3-11　创建res/layout-land

从res/layout目录复制activity_quiz.xml文件至res/layout-land目录。(如果在Android工具窗口看不到res/layout-land目录，请从Android视图切换到Project视图。完成后，记得切回。如果喜欢，你也可以直接使用文件管理器或终端应用复制粘贴文件。)

现在我们有了一个水平模式布局以及一个默认布局（竖直模式）。注意，两个布局文件的文件名必须相同，这样它们才能以同一个资源ID被引用。

为了与默认的布局文件相区别，还需修改水平模式布局文件。请参照图3-12进行相应修改。

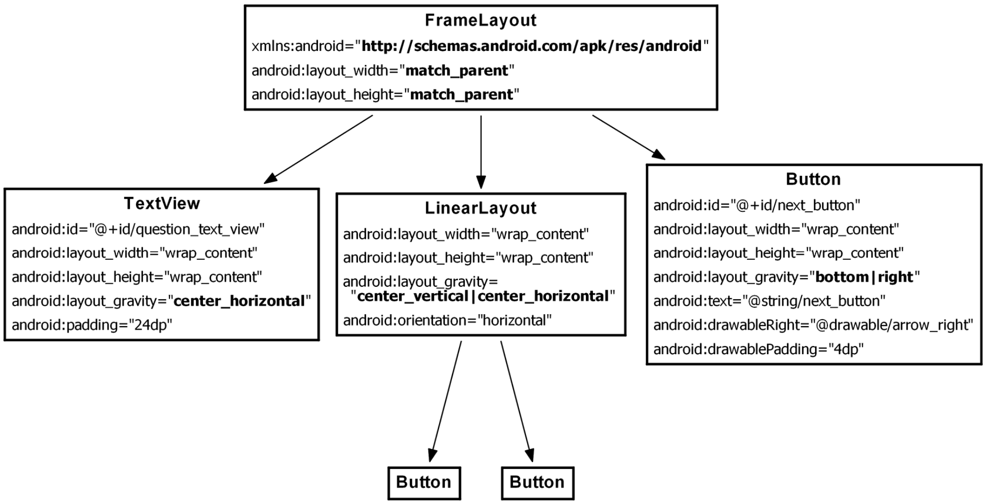


图3-12　备选的水平模式布局

如图3-12所示，FrameLayout替换了最上层的LinearLayout。FrameLayout是最简单的ViewGroup组件，它一概不管如何安排其子视图的位置。FrameLayout子视图的位置排列取决于它们各自的android:layout_gravity属性。

因而，TextView、LinearLayout和Button都需要一个android:layout_gravity属性。这里，LinearLayout里的Button子元素保持不变。

参照图3-12，打开layout-land/activity_quiz.xml文件修改。完成后可同代码清单3-4做对比检查。

代码清单3-4　水平模式布局修改（layout-land/activity_quiz.xml）

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="24dp" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center_horizontal"
        android:orientation="horizontal" >
      ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp"
        />

</LinearLayout>
</FrameLayout>
```


再次运行GeoQuiz应用。旋转设备至水平方位，查看新的布局界面，如图3-13所示。当然，这不仅是一个新的布局界面，也是一个新的QuizActivity。

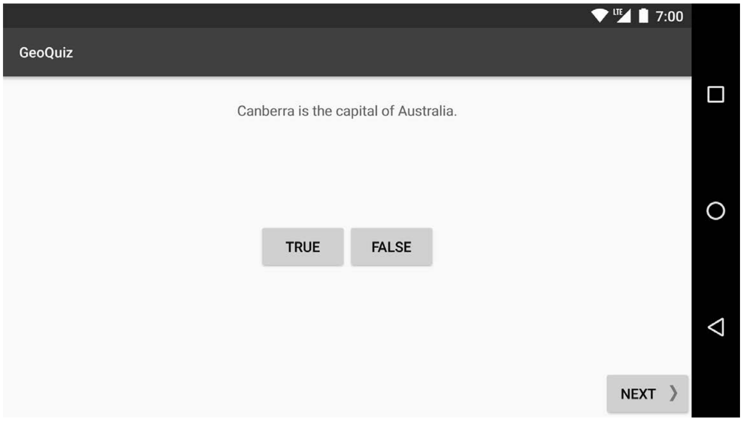


图3-13　处于水平方位的QuizActivity

设备旋转回竖直方位，可看到默认的布局界面以及另一个新的QuizActivity。

3.3　保存数据以应对设备旋转

适时使用备选资源，Android的这个点子不错，但是，设备旋转导致的activity销毁与新建有时也令人头疼。比如，设备旋转后，GeoQuiz应用将回到第一道题。

要修复这个缺陷，旋转后新建的QuizActivity需要知道mCurrentIndex变量的原值。显然，在设备运行中发生配置变更时，若设备旋转，需想个办法保存以前的数据。覆盖以下Activity方法就是一种解决方案：

```
protected void onSaveInstanceState(Bundle outState)
```

该方法通常在onStop()方法之前由系统调用，除非用户按后退键。（记住，按后退键就是告诉Android，activity用完了。随后，该activity就完全从内存中被抹掉，自然，也就没有必要为重建保存数据了。）

方法onSaveInstanceState(Bundle)的默认实现要求所有activity视图将自身状态数据保存在Bundle对象中。Bundle是存储字符串键与限定类型值之间映射关系（键–值对）的一种结构。

之前已用过Bundle，如下列代码所示，它作为参数传入onCreate(Bundle)方法：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
}
```

覆盖onCreate(Bundle)方法时，我们实际是在调用activity超类的onCreate(Bundle)方法，并传入收到的bundle。在超类代码实现里，通过取出保存的视图状态数据，activity的视图层级结构得以重建。

覆盖 onSaveInstanceState(Bundle)方法

可通过覆盖onSaveInstanceState(Bundle)方法，将一些数据保存在bundle中，然后在onCreate(Bundle)方法中取回这些数据。处理设备旋转问题时，将采用这种方式保存mCurrentIndex变量值。

首先，打开QuizActivity.java文件，新增一个常量作为将要存储在bundle中的键–值对的键，如代码清单3-5所示。

代码清单3-5 新增键–值对的键（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";

    private Button mTrueButton;
```

然后，覆盖onSaveInstanceState(Bundle)方法，以刚才新增的常量值作为键，将mCurrentIndex变量值保存到bundle中，如代码清单3-6所示。

代码清单3-6 覆盖onSaveInstanceState(Bundle)方法（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onPause() {
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        Log.i(TAG, "onSaveInstanceState");
        savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
    }

    @Override
    protected void onStop() {
        ...
    }
    ...
}
```

最后，在onCreate(Bundle)方法中确认是否成功获取该数值。如果获取成功，就将它赋值给变量mCurrentIndex，如代码清单3-7所示。

代码清单3-7 在onCreate(Bundle)方法中检查存储的bundle信息（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
```

```
        protected void onCreate(Bundle savedinstanceState) {
            super.onCreate(savedinstanceState);
            Log.d(TAG, "onCreate(Bundle) called ");
            setContentView(R.layout.activity_quiz);

            if (savedInstanceState != null) {
                mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
            }
            ...
        }
        ...
    }
```

运行GeoQuiz应用，单击NEXT按钮。现在，无论设备怎么旋转，新创建的`QuizActivity`都能记住当前正在回答的题目。

注意，在`Bundle`中存储和恢复的数据类型只能是基本类型（primitive type）以及可以实现`Serializable`或`Parcelable`接口的对象。在`Bundle`中保存定制类对象不是个好主意，因为你取回的对象可能已经没用了。比较好的做法是，通过其他方式保存定制类对象，而在`Bundle`中保存标识对象的基本类型数据。

3.4　再探 activity 生命周期

覆盖`onSaveInstanceState(Bundle)`方法并不仅仅用于处理与设备旋转相关的问题。用户离开当前activity用户界面，或Android需要回收内存时，activity也会被销毁。（例如，用户按了主屏幕键，然后播放视频或玩起游戏。）

基于用户体验考虑，Android从不会为了回收内存，而去销毁可见的activity（处于暂停或运行状态）。只有在调用过`onStop()`并执行完成后，activity才会被标为可销毁。

系统随时会销毁掉已停止的activity。不用担心数据丢失，activity停止时，会调用`onSaveInstanceState(Bundle)`方法的。所以，解决旋转数据丢失问题，就是抢在系统销毁activity之前保存数据。

保存在`onSaveInstanceState(Bundle)`的数据该如何幸免于难呢？调用该方法时，用户数据随即被保存在`Bundle`对象中，然后操作系统将`Bundle`对象放入activity记录中。

为便于理解activity记录，我们增加一个暂存状态（stashed state）到activity生命周期，如图3-14所示。

activity暂存后，`Activity`对象不再存在，但操作系统会将activity记录对象保存起来。这样，在需要恢复activity时，操作系统可以使用暂存的activity记录重新激活activity。

注意，activity进入暂存状态并不一定需要调用`onDestroy()`方法。不过，`onStop()`和`onSaveInstanceState(Bundle)`是两个可靠的方法（除非设备出现重大故障）。因而，常见的做法是，覆盖`onSaveInstanceState(Bundle)`方法，在`Bundle`对象中，保存当前activity的小的或暂存状态的数据；覆盖`onStop()`方法，保存永久性数据，如用户编辑的文字等。`onStop()`方法调用完，activity随时会被系统销毁，所以用它保存永久性数据。

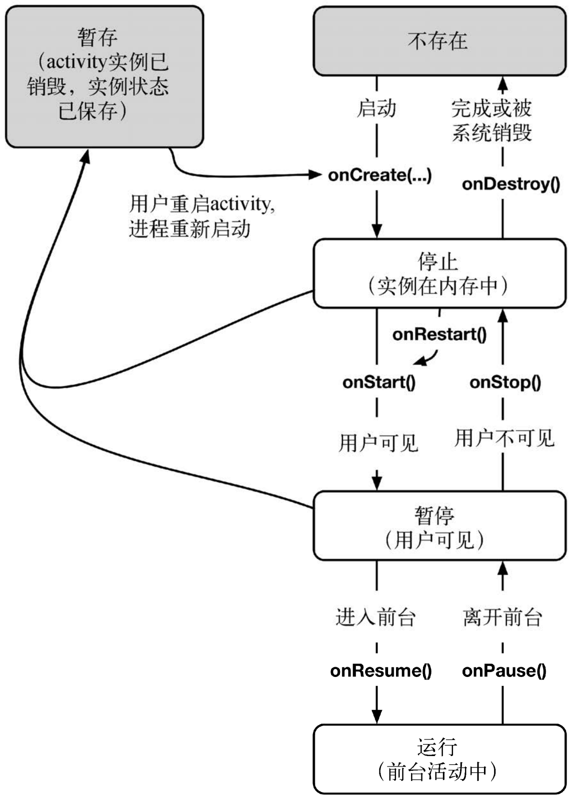


图3-14　完整的activity生命周期

那么暂存的activity记录到底可以保留多久？前面说过，用户按了后退键后，系统会彻底销毁当前的activity。此时，暂存的activity记录同时被清除。此外，系统重启的话，暂存的activity记录也会被清除。

3.5　深入学习：activity 内存清理现状

撰写本书时，低内存状态下，Android直接从内存清除整个应用进程，连带应用的所有activity。目前，Android还做不到只销毁单个activity。（Android应用都有自己的进程。进程的相关知识详见24.7节。）

相比其他进程，有前台（运行状态）或可见（暂停状态）activity的进程的优先级更高。需要释放资源时，Android系统的首选目标是低优先级进程。用户体验至上，理论上，操作系统不会杀掉带有可见activity的进程。当然出现重启或死机这样的大故障就难说了。（若真的出现这种情况，用户也没空深究具体是哪个应用被干掉了。）

覆盖`onSaveInstanceState(Bundle)`方法时，应测试activity状态是否如预期般正确保存和

恢复。旋转设备好测，低内存状况也好测。亲自试试吧。

在设备的应用列表中找到“设置”（Settings）。启动Settings，点击Developer options选项，找到并启用Don’t keep activities选项，如图3-15所示。

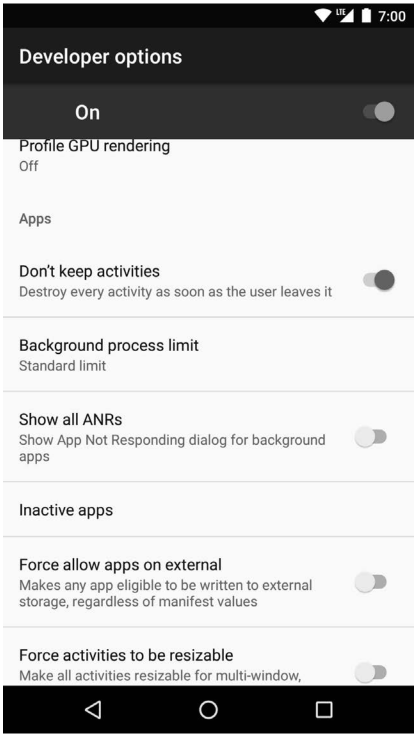


图3-15　启用Don’t keep activities选项

现在运行应用，单击主屏幕键。如前所述，点击主屏幕键会暂停并停止当前的activity。随后就像Android操作系统回收内存那样，停止的activity被系统销毁了。重新运行应用，验证activity状态是否如期得到保存。测试完毕，记得关闭Don’t keep activities选项，否则将导致系统和应用出现性能问题。

和单击主屏幕键不一样的是，单击后退键后，无论是否启用Don’t keep activities选项，系统总是会销毁当前的activity。单击后退键相当于告诉系统“用户不再需要使用当前的activity”。

3.6　深入学习：日志记录的级别与方法

使用`android.util.Log`类记录日志，不仅可以控制日志的内容，还可以控制用来区分信息重要程度的日志级别。Android支持如表3-2所示的五种日志级别。每一个级别对应一个`Log`类方法。要输出什么级别的日志，调用对应的`Log`类方法就可以了。

表3-2 日志级别与方法

日志级别	方 法	说 明
ERROR	Log.e(...)	错误
WARNING	Log.w(...)	警告
INFO	Log.i(...)	信息型消息
DEBUG	Log.d(...)	调试输出（可能被过滤掉）
VERBOSE	Log.v(...)	仅用于开发

需要说明的是，所有的日志记录方法都有两种参数签名：string类型的tag参数和msg参数；除tag和msg参数外再加上Throwable实例参数。附加的Throwable实例参数为应用抛出异常时记录异常信息提供了方便。代码清单3-8展示了两种方法不同参数签名的使用实例。对于输出的日志信息，可使用常用的Java字符串连接操作拼接出需要的信息，或者使用String.format对输出日志信息进行格式化操作，以满足个性化的使用要求。

代码清单3-8 Android的各种日志记录方式

```
// Log a message at "debug" log level
Log.d(TAG, "Current question index: " + mCurrentIndex);

Question question;
try {
    question = mQuestionBank[mCurrentIndex];
} catch (ArrayIndexOutOfBoundsException ex) {
    // Log a message at "error" log level, along with an exception stack trace
    Log.e(TAG, "Index was out of bounds", ex);
}
```

3.7 挑战练习：禁止一题多答

用户答完某道题，就禁掉那道题对应的按钮，防止用户一题多答。

3.8 挑战练习：评分

用户答完全部题后，显示一个toast消息，给出百分比形式的评分。

第 4 章

Android应用的调试

本章将讲解如何处理应用的bug，同时也会介绍如何使用LogCat、Android Lint以及Android Studio内置的代码调试器。

为练习调试，我们先搞点破坏。打开QuizActivity.java文件，在`onCreate(Bundle)`方法中，注释掉`mQuestionTextView`变量赋值的那行代码，如代码清单4-1所示。

代码清单4-1 注释掉一行关键代码（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);

    if (savedinstanceState != null) {
        mCurrentindex = savedinstanceState.getint(KEY_iNDEX, 0);
    }

    // mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        ...
    });
    ...
}
```

运行GeoQuiz应用，看看会发生什么。图4-1是应用崩溃后的消息提示画面。不同Android版本的消息提示略有差异，但本质上都是一个意思。

显然，我们知道应用为何崩溃。假如不知道的话，接下来的全新视角或许能帮助解决问题。

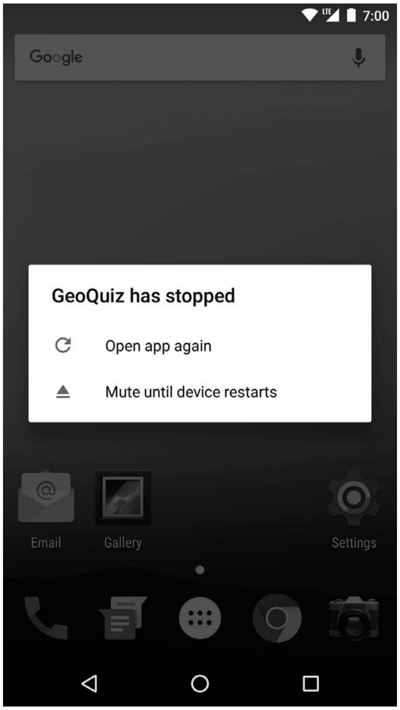


图4-1　GeoQuiz应用崩溃了

4.1　异常与栈跟踪

为了方便查看，展开Android Monitor工具窗口。上下滑动LogCat窗口滚动条，应该会看到整片红色的异常或错误信息，如图4-2所示。这就是标准的`AndroidRuntime`异常信息报告。

如果看不到，可试着选择LogCat的No Filters过滤项。另外，如果觉得信息太多，看不过来，还可以调整Log Level为Error，让系统只输出严重问题日志。还可以使用搜索功能，比如搜“FATAL EXCEPTION”，就能直接定位到崩溃异常。

该异常报告首先给出最高层级的异常及其栈跟踪，然后是导致该异常的异常及其栈跟踪。如此不断追溯，直到找到一个没有原因的异常。

在我们编写的大部分代码中，最后一个没给出原因的异常往往就是关注点。这里，没有原因的异常是`java.lang.NullPointerException`。紧接着该异常语句的一行就是其栈跟踪信息的第一行。从该行可以看出发生异常的类和方法以及它所在的源文件及代码行号。单击蓝色链接，Android Studio会自动跳转到源代码的对应代码行。

Android Studio定位的这行代码是`mQuestionTextView`变量在`updateQuestion()`方法中的首次使用。名为`NullPointerException`的异常暗示了问题所在，即变量没有初始化。

```
09-03 12:44:08.523    5458-5458/? E/AndroidRuntime﹕ FATAL EXCEPTION: main
    Process: com.bignerdranch.android.geoquiz, PID: 5458
    java.lang.RuntimeException: Unable to start activity ComponentInfo{com.bignerdranch.android.ge
            at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2184)
            at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
            at android.app.ActivityThread.access$800(ActivityThread.java:135)
            at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
            at android.os.Handler.dispatchMessage(Handler.java:102)
            at android.os.Looper.loop(Looper.java:136)
            at android.app.ActivityThread.main(ActivityThread.java:5001)
            at java.lang.reflect.Method.invokeNative(Native Method) <1 internal calls>
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
            at dalvik.system.NativeStart.main(Native Method)
     Caused by: java.lang.NullPointerException
            at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:35)
            at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:90)
            at android.app.Activity.performCreate(Activity.java:5231)
            at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1087)
            at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2148)
            at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
            at android.app.ActivityThread.access$800(ActivityThread.java:135)
            at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
            at android.os.Handler.dispatchMessage(Handler.java:102)
            at android.os.Looper.loop(Looper.java:136)
            at android.app.ActivityThread.main(ActivityThread.java:5001)
            at java.lang.reflect.Method.invokeNative(Native Method)
            at java.lang.reflect.Method.invoke(Method.java:515) <3 more...>
```

图4-2　LogCat中的异常与栈跟踪

为修正该问题，取消对变量`mQuestionTextView`赋值语句的注释。

碰到运行异常时，记得在LogCat中寻找最后一个异常及其栈跟踪的第一行（对应着源代码）。这里是问题发生的地方，也是寻找解决方案的最佳起点。

如果发生应用崩溃的设备没有与计算机连接，日志信息也不会全部丢失。设备会将最近的日志保存到日志文件中。日志文件的内容长度及保留的时间取决于具体的设备，不过，获取十分钟之内产生的日志信息通常是有保证的。只要将设备连上计算机，在Devices视图里选择所用设备，LogCat会自动打开并显示日志文件保存的内容。

4.1.1　诊断应用异常

即使出了问题，应用也不一定会崩溃。某些时候，应用只是出现了运行异常。例如，每次单击NEXT按钮时，应用都毫无反应。这就是一个非崩溃型的应用运行异常。

在QuizActivity.java中，修改`mNextButton`监听器代码，注释掉`mCurrentIndex`变量递增的语句，如代码清单4-2所示。

代码清单4-2　注释掉一行关键代码（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
```

```
            }
        });
        ...
    }
```

运行GeoQuiz应用，点击NEXT按钮。可以看到，应用无响应。

这个问题要比上一个棘手。它没有抛出异常，所以，解决起来不像前面跟踪追溯并消除异常那么简单。有了前面的经验，这里可以推测出导致该问题的两个因素：

- ❑ `mCurrentIndex`变量值没有改变；
- ❑ `updateQuestion()`方法没被调用。

如果实在没有头绪，则需要设法跟踪并找出问题所在。在接下来的几小节里，我们将学习两种跟踪问题的方法：

- ❑ 记录栈跟踪的诊断性日志；
- ❑ 利用调试器设置断点调试。

4.1.2 记录栈跟踪日志

在`QuizActivity`中，为`updateQuestion()`方法添加日志输出语句，如代码清单4-3所示。

代码清单4-3 方便实用的调试方式（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    private void updateQuestion() {
        Log.d(TAG, "Updating question text ", new Exception());
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }
```

如同前面AndroidRuntime的异常，`Log.d(String, String, Throwable)`方法记录并输出整个栈跟踪日志。这样，就可以很容易看出`updateQuestion()`方法在哪些地方被调用了。

作为参数传入`Log.d(String, String, Throwable)`方法的异常不一定就是已捕获的抛出异常。可以创建一个全新的`Exception`，把它作为不抛出的异常对象传入该方法。借此，我们得到异常发生位置的记录报告。

运行GeoQuiz应用，点击NEXT按钮，然后在LogCat中查看输出结果，如图4-3所示。

```
09-04 12:47:37.733  30612-30612/com.bignerdranch.android.geoquiz D/QuizActivity: Updating question text
    java.lang.Exception
            at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:34)
            at com.bignerdranch.android.geoquiz.QuizActivity.access$100(QuizActivity.java:12)
            at com.bignerdranch.android.geoquiz.QuizActivity$3.onClick(QuizActivity.java:83)
            at android.view.View.performClick(View.java:4438)
            at android.view.View$PerformClick.run(View.java:18422)
            at android.os.Handler.handleCallback(Handler.java:733)
            at android.os.Handler.dispatchMessage(Handler.java:95)
            at android.os.Looper.loop(Looper.java:136)
            at android.app.ActivityThread.main(ActivityThread.java:5001)
            at java.lang.reflect.Method.invokeNative(Native Method) <1 internal calls>
            at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
            at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
            at dalvik.system.NativeStart.main(Native Method)
```

图4-3 输出结果

栈跟踪日志的第一行即调用异常记录方法的地方。紧接着的两行表明，updateQuestion()方法是在onClick(...)实现方法里被调用的。点击该行链接跳转至注释掉的问题索引递增代码行。暂时不要修正，下一节还会使用设置断点调试的方法重新查找该问题。

记录栈跟踪日志虽然是个强大的工具，但也存在缺陷。比如，大量的日志输出很容易导致LogCat窗口信息混乱难读。此外，通过阅读详细直白的栈跟踪日志并分析代码意图，竞争对手可以轻易剽窃我们的创意。

另一方面，既然有可能从栈跟踪日志看出代码的真实意图，在网站http://stackoverflow.com或者论坛http://forums.bignerdranch.com上求助时，附上一段栈跟踪日志往往有助于解决问题。如果需要这样做，你可以直接从LogCat中复制并粘贴日志内容。

继续学习之前，先删除日志记录代码，如代码清单4-4所示。

代码清单4-4　再见，老朋友（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    private void updateQuestion() {
        Log.d(TAG, "Updating question text", new Exception());
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }
```

4.1.3　设置断点

要使用Android Studio自带调试器调试上一节中的问题，首先要在updateQuestion()方法中设置断点，以确认该方法是否被调用。断点会在断点设置行的前一行处停止代码执行，然后我们可以逐行检查代码，看看接下来到底发生了什么。

在QuizActivity.java文件中，找到updateQuestion()方法，点击第一行代码左边的灰色栏区域。可以看到，灰色栏上出现了一个红色圆点。这就是已设置的一处断点，如图4-4所示。

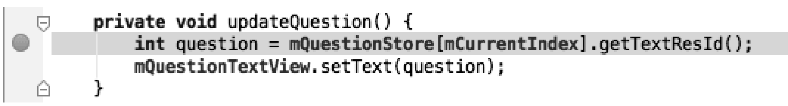


图4-4　已设置的一处断点

为启用代码调试器并触发已设置的断点，我们需要调试运行而不是直接运行应用。要调试运行应用，单击Run按钮旁边的Debug按钮，或选择Run → Debug 'app' 菜单项。设备会报告说正在等待调试器加载，然后继续运行。

应用启动并加载调试器运行后，就会暂停。应用首先调用QuizActivity.onCreate(Bundle)方法，该方法又调用updateQuestion()方法，然后触发断点。

如图4-5所示，QuizActivity.java代码已经在代码编辑区打开了，断点设置所在行的代码也被加亮显示了。应用在断点处停止运行。

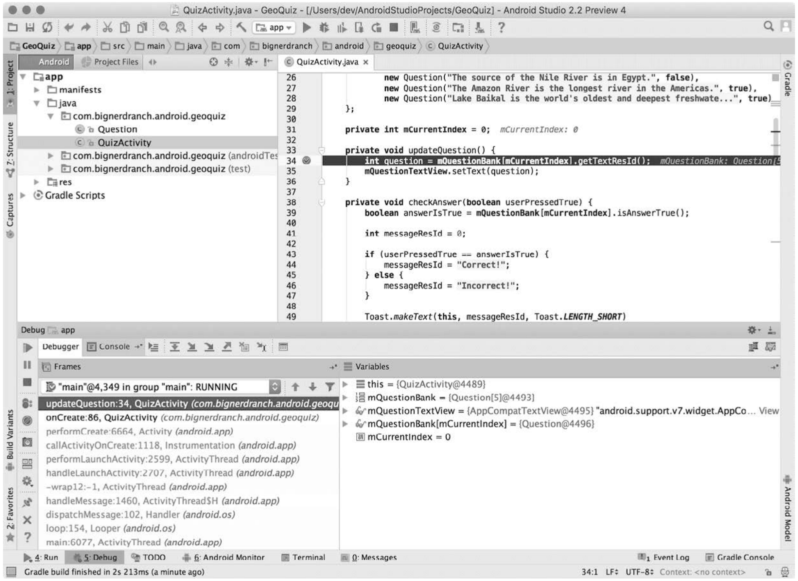


图4-5 代码在断点处停止执行

这时，由Frames和Variables视图组成的Debug工具窗口出现在屏幕底部，如图4-6所示。



图4-6 代码调试视图

使用视图顶部的箭头按钮可单步执行应用代码。从栈列表可以看出，`updateQuestion()`方法已经在`onCreate(Bundle)`方法中被调用了。不过，我们关心的是NEXT按钮被点击后的行为。因此，单击“继续运行”按钮。然后，再次点击GeoQuiz中的NEXT按钮，观察断点是否被激活（应该会激活）。

既然程序执行停在了断点处，就可以趁机看看其他视图。变量视图（Variables）可以让我们观察到程序中各对象的值。应该可以看到在`QuizActivity`中创建的变量，以及一个特别的`this`

变量值（QuizActivity本身）。

展开this变量后可看到很多变量。它们是QuizActivity类的Activity超类、Activity超类的超类（一直追溯到继承树顶端）的全部变量。

我们只需关心mCurrentIndex变量值。在变量视图里滚动查看并找到mCurrentIndex。显然，它现在的值为0。

代码看上去没问题。为继续追查，需跳出当前方法。单击“单步执行跳出”按钮。

查看代码编辑视图，我们现在跳到了mNextButton的OnClickListener方法，正好是在updateQuestion()方法被调用之后。真是相当方便的调试，问题解决了。

接下来就是代码修正。不过，要修改代码，必须先停止调试应用。停止调试有以下两种方式：

- 停止程序，单击图4-6所示的“停止”按钮；
- 断开调试器，单击图4-6所示的“关闭”按钮。

回到代码编辑区，在OnClickListener方法中，取消对mCurrentIndex语句的注释，如代码清单4-5所示。

代码清单4-5　取消代码注释（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
        }
    });
    ...
}
```

至此，我们尝试了两种不同的代码跟踪调试方法：

- 记录栈跟踪的诊断性日志；
- 利用调试器设置断点调试。

哪种方式更好？没有肯定的答案，它们各有所长。实际体验之后，或许各有所爱吧。

栈跟踪记录的优点是，在同一日志记录中可以看到多处栈跟踪信息；缺点是，必须学习如何添加日志记录方法，重新编译、运行应用并跟踪排查应用问题。相对而言，代码调试的方法更为方便。应用以调试模式运行后，可在应用运行的同时，在不同的地方设置断点，寻找解决问题的线索。

4.1.4　使用异常断点

前面介绍的调试方法还不够用？那就试试使用调试器来捕捉异常吧。在QuizActivity.java中，

注释掉一行代码，让应用崩溃，如代码清单4-6所示。

代码清单4-6 使GeoQuiz再次崩溃（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
    ...
    // mNextButton = (Button) findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
        }
    });
    ...
}
```

选择Run → View Breakpoints...菜单项调出异常断点设置窗口，如图4-7所示。

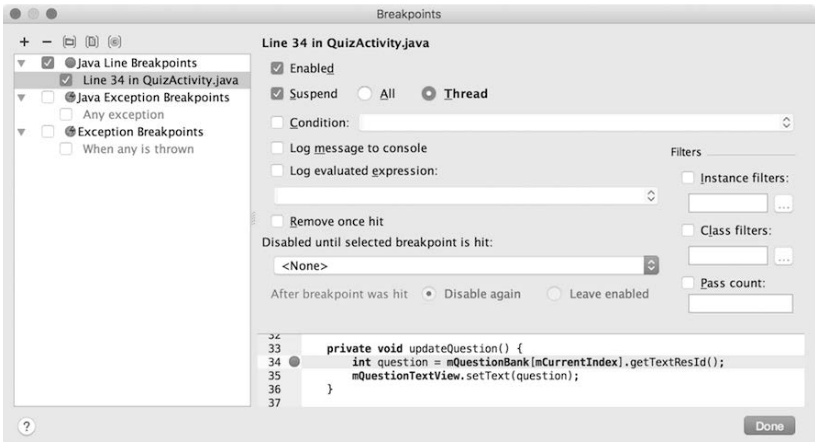


图4-7 设置异常断点

可以看到，当前设置的断点都显示在左边窗口，选中先前设置的断点，点击删除按钮（-）删除。

可以使用该对话窗口设置新断点。这样，无论任何时候，只要应用抛出异常就可以触发该断点。如果需要，可限制断点仅针对未捕获的异常生效，也可以设置为对两种类型的异常（未捕获的和已捕获的异常）都生效。

单击新增断点按钮（+）设置一个新断点。选择下拉列表中的Java Exception Breakpoints选项。接下来选择要捕捉的异常类型。输入`RuntimeException`，按提示选择RuntimeException（java.lang）。`RuntimeException`是`NullPointerException`、`ClassCastException`及其他常见异常的超类，因此该设置基本适用于所有异常。

点击Done按钮完成设置。调试GeoQuiz应用。这次，调试器很快就定位到异常抛出的代码行。真是太棒了。

异常断点影响极大，建议及时清除那些不需要的断点。否则，在调试的时候，如果我们不关心的一些系统框架代码或者其他地方发生异常，就会触发先前设置的断点。继续学习之前，删除刚才设置的断点。

取消对QuizActivity.java的代码注释，让GeoQuiz应用恢复正常。

4.2　Android 特有的调试工具

总体来讲，Android应用调试和Java应用调试没什么两样。然而，Android也有其特有的应用调试场景，如应用资源问题。显然，Java编译器并不擅长处理此类问题。

4.2.1　使用 Android Lint

该是Android Lint发挥作用的时候了。Android Lint是Android应用代码的静态分析器（static analyzer）。作为一个特殊程序，它能在不运行代码的情况下检查代码错误。凭着对Android框架的熟练掌握，Android Lint能深入检查代码，找出编译器无法发现的问题。在大多数情况下，Android Lint检查出的问题都值得重视。

在第6章，我们会看到Android Lint对设备兼容问题的警告。此外，Android Lint能够检查定义在XML文件中的对象类型。在QuizActivity.java中，人为制造一处错误，如代码清单4-7所示。

代码清单4-7　不匹配的对象类型（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);
    ...
    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton = (Button)findViewById(R.id.question_text_view);
    ...
}
```

因为使用了错误的资源ID，所以代码运行时，会尝试把`TextView`转换为`Button`类型，这会导致类型转换错误。显然，Java编译器无能为力，而Android Lint就能捕获到该错误。可以看到Lint立即高亮显示该行代码，指出问题。

假如想主动查看项目中的所有潜在问题，可以选择Analyze → Inspect Code...菜单项手动运行Lint。在被问及检查项目的哪部分时，选择Whole project。Android Studio会立即运行Lint和其他一些静态分析器开始分析代码。

检查完毕，所有的潜在问题会按类别列出。展开Android Lint类别，可看到具体的Lint信息，

如图4-8所示。

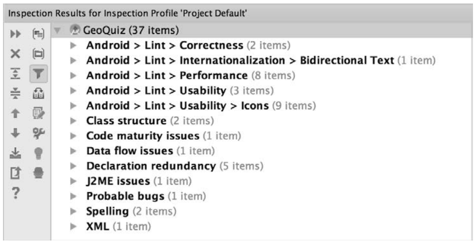


图4-8 Lint警告信息

继续展开还可以看到更加详细的信息，包括问题发生的地方。

Mismatched view type错误是我们人为制造的。现在，恢复代码如初，如代码清单4-8所示。

代码清单4-8 修正类型不匹配的代码错误（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.question_text_view);
    mTrueButton = (Button)findViewById(R.id.true_button);
    ...
}
```

最后，重新运行GeoQuiz应用，确认已恢复。

4.2.2 R 类的问题

对于引用还未添加的资源，或者删除仍被引用的资源而导致的编译错误，我们已经很熟悉了。通常，在添加资源或删除引用后重新保存文件，Android Studio会准确无误地重新编译项目。

不过，资源编译错误有时会一直存在或莫名其妙地出现。如遇这种情况，请尝试如下操作。

- 重新检查资源文件中XML文件的有效性

  如果最近一次编译时未生成R.java文件，项目中资源引用的地方都会出错。通常，这是由某个布局XML文件中的拼写错误引起的。既然布局XML文件有时无法得到有效校验，拼写错误自然也就难以发现了。修正找到的错误并重新保存XML文件，Android Studio会生成新的R.java文件。

- 清理项目

  选择Build → Clean Project菜单项。Android Studio会重新编译整个项目，消除错误。建议经常做深度项目清理。

- 使用Gradle同步项目

  如果修改了build.gradle配置文件，就需要同步更新项目的编译设置。选择Tools → Android → Sync Project with Gradle Files菜单项，Android Studio会使用正确的项目设置重新编译项目。这会解决Gradle配置变更带来的问题。

- 运行Android Lint

  仔细查看Lint警告信息，没准就会有新发现。

如果仍有资源相关问题或其他问题，建议仔细阅读错误提示并检查布局文件。慌乱时往往找不出问题。不妨休息冷静一下，再重新查看Android Lint报告的错误和警告，或许就能找出代码错误或拼写输入错误。

如果上述操作无法解决问题，或遇到其他Android Studio使用问题，还可以访问网站http://stackoverflow.com或本书论坛http://forums.bignerdranch.com求助。

4.3　挑战练习：探索布局检查器

为了调试布局文件，可使用布局检查器以交互的方式检查布局文件，研究它是如何在屏幕上渲染显示的。要使用布局检查器，首先在模拟器上启动GeoQuiz应用，然后在Android Monitor工具窗口点击最左边的布局检查器按钮，如图4-9所示。布局检查器激活后，点击布局检查器视图里的元素，就可以查看布局属性了。

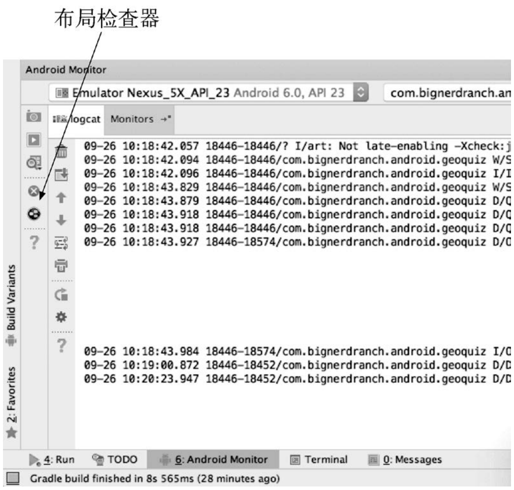


图4-9　布局检查器按钮

4.4　挑战练习：探索内存分配跟踪

针对应用里内存分配的频率和次数，内存分配跟踪器能给出详细的跟踪报告。这大大方便了

应用的性能优化。在Android Monitor工具窗口，点击内存分配跟踪器按钮启动它，如图4-10所示。

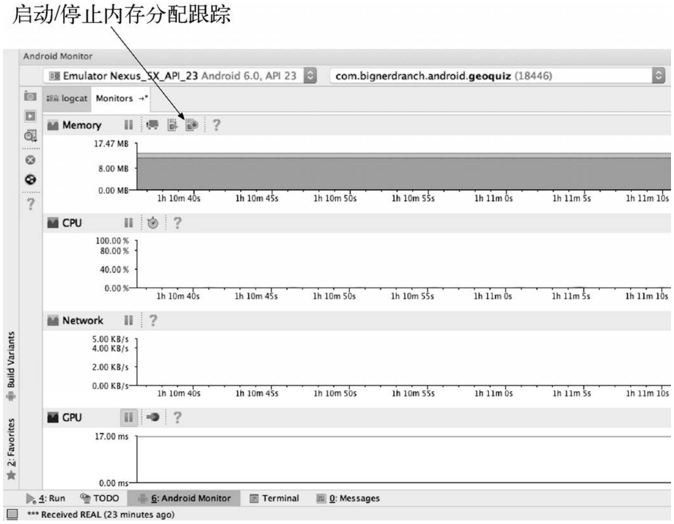


图4-10　启动内存分配跟踪器

随后，你在前台操作应用，后台就开始记录内存分配状况。一旦找到你想优化的点，就可以再次点击内存分配跟踪器按钮停止。内存分配报告随之展现，如图4-11所示。

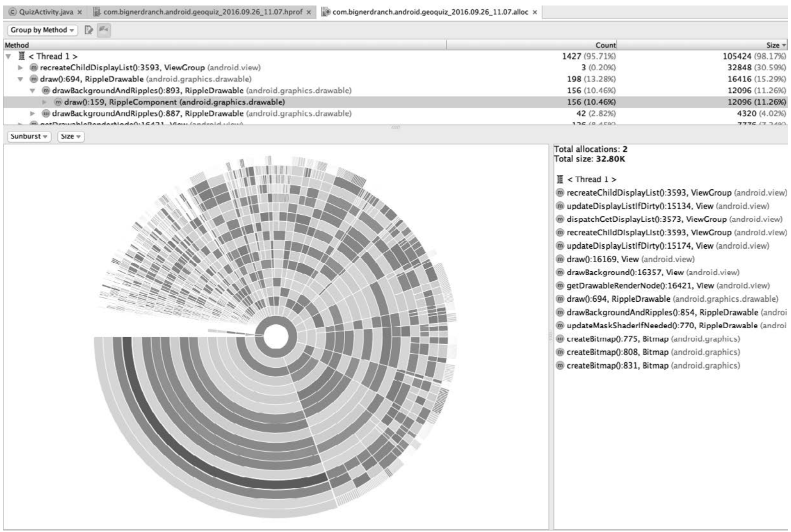


图4-11　内存分配报告

内存分配报告会列出内存分配发生的次数以及字节大小。在工具顶端选择你需要的报表类型，报表展现形式可以是表，也可以是图。

第 5 章

第二个activity

本章，我们为GeoQuiz应用添加第二个activity。一个activity控制一屏信息，新activity将带来第二个用户界面，方便用户偷看当前问题的答案，如图5-1所示。

如果用户选择先看答案，然后返回`QuizActivity`答题，则会收到一条信息，如图5-2所示。



图5-1　`CheatActivity`提供了偷看答案的机会

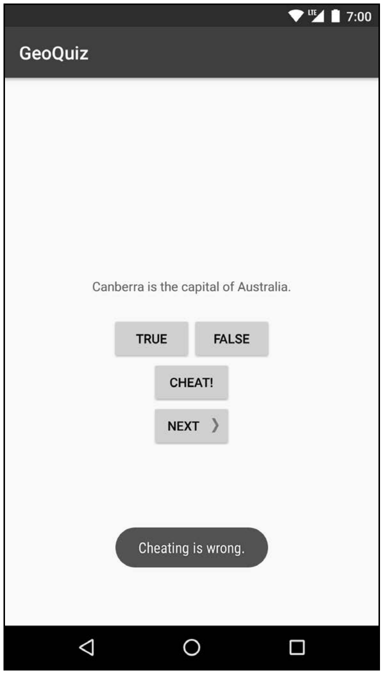


图5-2　有没有偷看答案，`QuizActivity`都知道

完成GeoQuiz应用的升级，我们可以学到以下知识点。

- 创建新的activity及配套布局。
- 从一个activity中启动另一个activity。所谓*启动*activity，就是请求Android系统创建新的activity实例并调用其`onCreate(Bundle)`方法。
- 在父activity（启动方）与子activity（被启动方）间传递数据。

5.1　创建第二个 activity

要创建新的activity，有好多繁杂工作要做。好在Android Studio有省事省心的新建activity向导。

省不省事，稍后便知。现在先打开strings.xml文件，添加本章要用的所有字符串资源，如代码清单5-1所示。

代码清单5-1　添加字符串资源（strings.xml）

```
<resources>
    ...
    <string name=" Incorrect_toast"> Incorrect!</string>
    <string name="warning_text">Are you sure you want to do this?</string>
    <string name="show_answer_button">Show Answer</string>
    <string name="cheat_button">Cheat!</string>
    <string name="judgment_toast">Cheating is wrong.</string>

</resources>
```

5.1.1　创建新的 activity

创建新的activity至少涉及三个文件：Java类、XML布局和应用的manifest文件。这三个文件关联密切，搞错了就是灾难。因此，强烈建议使用Android Studio的新建activity向导功能。

在项目工具窗口中，右键单击com.bignerdranch.android.geoquiz包，选择New → Activity → Empty Activity菜单项启动新建activity向导，如图5-3所示。

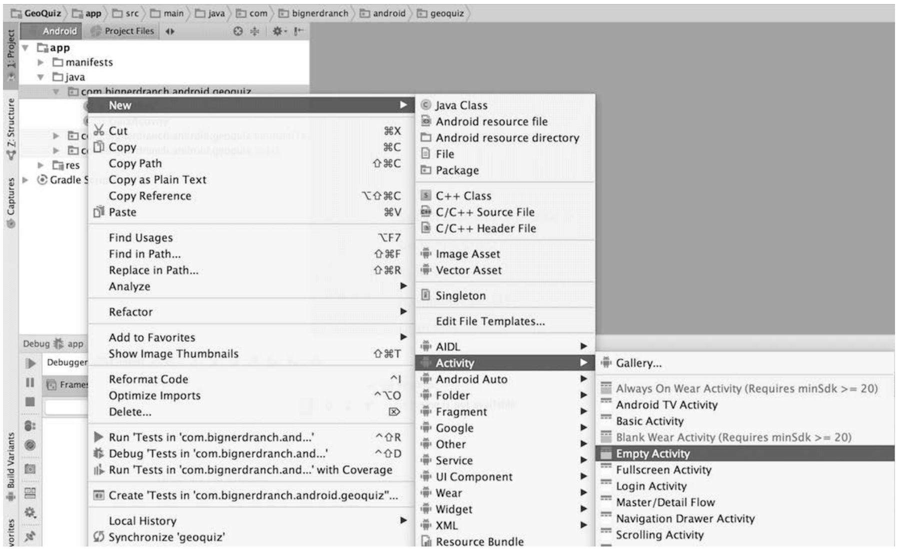


图5-3　新建activity向导菜单

在随后弹出的对话框中，Activity Name处输入CheatActivity，如图5-4所示。这是Activity子类的名字。可以看到，Layout Name自动赋值为activity_cheat。这是向导为布局文件创建的基本名称。

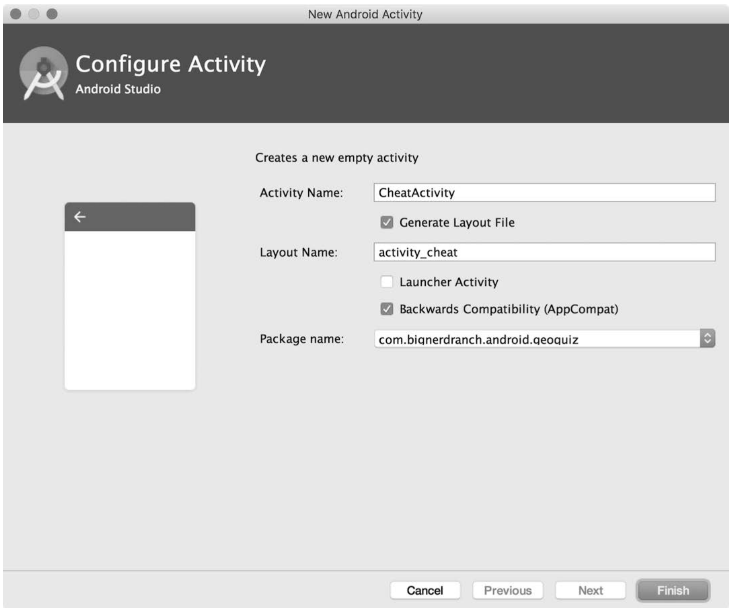


图5-4　新的空activity向导

由于包名决定CheatActivity.java文件存放的位置，再看看包名是否符合要求。最后，保持其他默认设置不变，点击Finish按钮见证向导强大的功能。

接下来的任务是设计美观的用户界面。本章开头的截图是CheatActivity视图完成后的样子。组成视图的组件定义如图5-5所示。

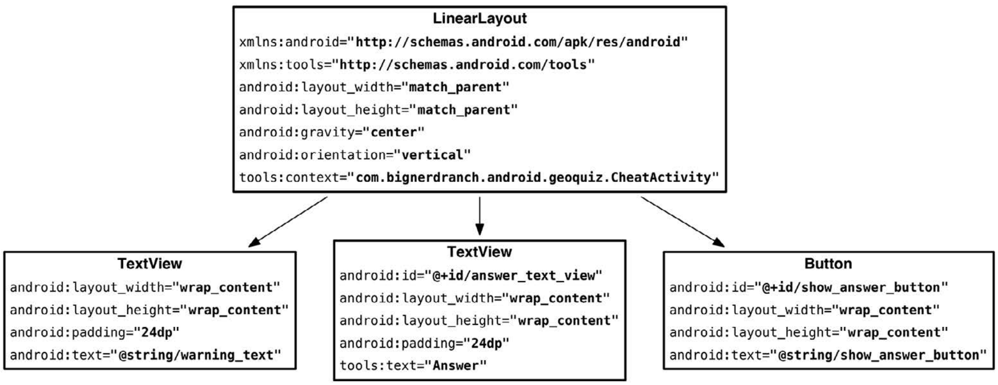


图5-5　组件定义示意图

Android Studio应该已经打开layout目录中的activity_cheat.xml。如果没有，请打开它并切换至文字视图模式。

参照图5-5创建布局XML文件，依次以LinearLayout组件替换样例布局。第9章以后，我们将不再给出大段的XML代码，而只会给出类似图5-5的布局组件示意图。所以，最好现在就习惯参照图示创建布局XML文件。完成后，记得对照代码清单5-2核查。

代码清单5-2　第二个activity的布局组件定义（activity_cheat.xml）

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:tools="http://schemas.android.com/tools"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              android:gravity="center"
              tools:context="com.bignerdranch.android.geoquiz.CheatActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/warning_text"/>

    <TextView
        android:id="@+id/answer_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        tools:text="Answer"/>

    <Button
        android:id="@+id/show_answer_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/show_answer_button"/>

</LinearLayout>
```

注意用于显示答案的TextView组件，它的tools和tools:text属性的命名空间比较特别。该命名空间可以覆盖某个组件的任何属性。这样，就可以在Android Studio预览中看到效果。TextView有text属性，可以为它提供初始值，因而在应用运行前就知道它大概的样子。而在应用运行时，Answer文字不会显示出来。真的很方便！

虽然没有创建供设备处于水平方位时使用的布局文件，不过，借助开发工具，我们可以预览默认布局在水平方位时的显示效果。

在预览工具窗口中，找到预览界面上方工具栏里一个画着旋转设备的按钮（带弧形蓝色箭头）。单击该按钮切换布局预览方位，如图5-6所示。

5

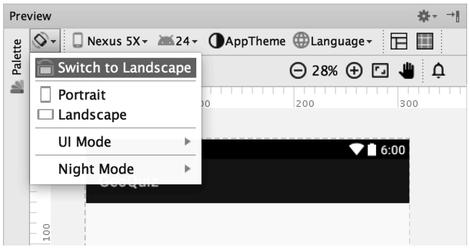


图5-6　水平方位预览布局（activity_cheat.xml）

可以看到，默认布局在竖直方位与水平方位下效果都不错。布局搞定了，接着是创建新的activity子类。

5.1.2　创建新的 activity 子类

在项目工具窗口中，找到com.bignerdranch.android.geoquiz类包，打开CheatActivity.java文件。

当前，`CheatActivity`类已有`onCreate(Bundle)`方法的默认实现，用来将activity_cheat.xml文件中的布局资源ID传递给`setContentView(...)`方法。

`CheatActivity`类的`onCreate(Bundle)`方法还有更多的事情要做。现在，先一起来看看新建activity向导自动完成的另一件事：manifest配置文件中的`CheatActivity`声明。

5.1.3　在 manifest 配置文件中声明 activity

manifest配置文件是一个包含元数据的XML文件，用来向Android操作系统描述应用。该文件总是以AndroidManifest.xml命名，可在项目的app/manifests目录中找到它。

在项目工具窗口中，找到并打开AndroidManifest.xml。还可使用Android Studio的快速打开文件功能：使用Command+Shift+O（或Ctrl+Shift+N）快捷键，呼出快速打开对话框，利用提示功能或直接输入目标文件名，按回车键打开。

应用的所有activity都必须在manifest配置文件中声明，这样操作系统才能够找到它们。

创建`QuizActivity`时，因使用了新建应用向导，向导已自动完成声明工作。同样，新建activity向导也自动声明了`CheatActivity`，如代码清单5-3灰底部分所示。

代码清单5-3　在manifest配置文件中声明`CheatActivity`（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.geoquiz" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
```

```
        <activity android:name=".QuizActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".CheatActivity">
        </activity>
    </application>

</manifest>
```


这里的`android:name`属性是必需的。属性值前面的`.`告诉操作系统：activity类文件位于manifest配置文件头部包属性值指定的包路径下。

`android:name`属性值也可以设置成完整的包路径，如`android:name="com.bignerdranch.android.geoquiz.CheatActivity"`，这与代码清单5-3里的写法效果相同。

manifest配置文件里还有很多有趣的东西。不过，现在还是先集中精力搞定`CheatActivity`的配置和运行。在后续章节中，我们还将学习到更多有关manifest配置文件的知识。

5.1.4 为 QuizActivity 添加 CHEAT 按钮

按照开发设想，用户在`QuizActivity`用户界面上点击某个按钮，应用立即创建`CheatActivity`实例，并显示其用户界面。这就需要在layout/activity_quiz.xml和layout-land/activity_quiz.xml布局文件中定义新按钮。

在默认的垂直布局中，添加新按钮定义并设置其为根`LinearLayout`的直接子类。新按钮应该定义在NEXT按钮之前，如代码清单5-4所示。

代码清单5-4 在默认布局中添加CHEAT按钮（layout/activity_quiz.xml）

```
      ...
    </LinearLayout>

    <Button
        android:id="@+id/cheat_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheat_button"/>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp"/>

</LinearLayout>
```

在水平布局模式中，新按钮定义在根FrameLayout的底部居中位置，如代码清单5-5所示。

代码清单5-5　在水平布局中添加CHEAT按钮（layout-land/activity_quiz.xml）

```
      ...
    </LinearLayout>

    <Button
        android:id="@+id/cheat_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|center_horizontal "
        android:text="@string/cheat_button" />

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp" />

</FrameLayout>
```

重新打开QuizActivity.java文件，添加新按钮变量以及资源引用代码。最后为CHEAT按钮添加View.onClickListener监听器代码存根，如代码清单5-6所示。

代码清单5-6　启用CHEAT按钮（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    private Button mNextButton;
    private Button mCheatButton;
    private TextView mQuestionTextView;
    ...
    @Override
    protected void onCreate(Bundle savedinstanceState) {
        ...
        mNextButton = (Button) findViewByid(R.id.next_button);
        mNextButton.setOnClicklistener(new View.OnClicklistener() {
            @Override
            public void onClick(View v) {
                mCurrentindex = (mCurrentindex + 1) % mQuestionBank.length;
                updateQuestion();
            }
        });

        mCheatButton = (Button)findViewById(R.id.cheat_button);
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Start CheatActivity
            }
        });
```

```
        updateQuestion();
    }
  ...
}
```

准备工作完成了，下面来学习如何启动CheatActivity。

5.2 启动 activity

一个activity启动另一个activity最简单的方式是使用startActivity方法：

```
public void startActivity(Intent intent)
```


你也许会想当然地认为，startActivity(Intent)方法是一个静态方法，启动activity就是调用Activity子类的该方法。实际并非如此。activity调用startActivity(Intent)方法时，调用请求实际发给了操作系统。

准确地说，调用请求发送给了操作系统的ActivityManager。ActivityManager负责创建Activity实例并调用其onCreate(Bundle)方法，如图5-7所示。

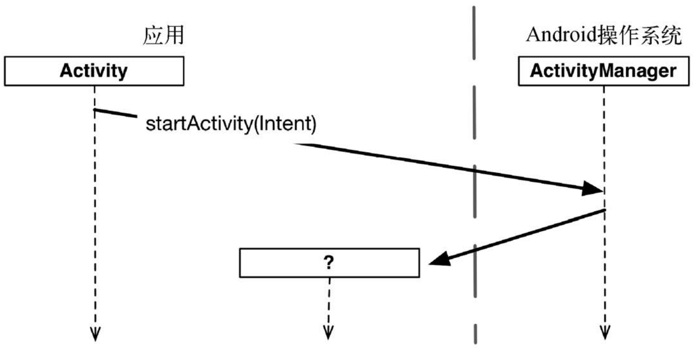


图5-7 启动activity

ActivityManager该启动哪个activity呢？答案就在于传入startActivity(Intent)方法的Intent参数。

基于 intent 的通信

intent对象是component用来与操作系统通信的一种媒介工具。目前为止，我们唯一见过的component就是activity。实际上还有其他一些component：service、broadcast receiver以及content provider。

intent是一种多用途通信工具。Intent类有多个构造方法，能满足不同的使用需求。

在GeoQuiz应用中，intent用来告诉ActivityManager该启动哪个activity，因此可使用以下构造方法：

```
public Intent(Context packageContext, Class<?> cls)
```

传入该方法的`Class`类型参数告诉`ActivityManager`应该启动哪个activity；`Context`参数告诉`ActivityManager`在哪里可以找到它，如图5-8所示。

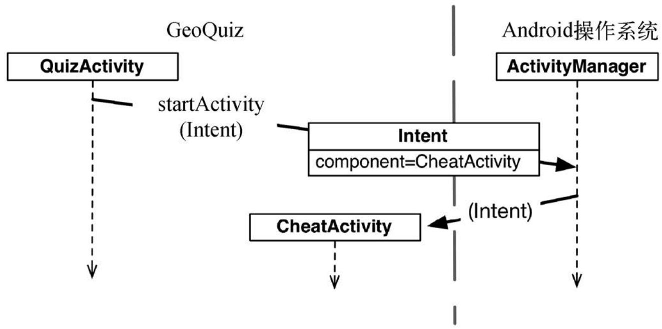


图5-8　intent：`ActivityManager`的信使

在`mCheatButton`的监听器代码中，创建包含`CheatActivity`类的`Intent`实例，然后将其传入`startActivity(Intent)`方法，如代码清单5-7所示。

代码清单5-7　启动`CheatActivity`（QuizActivity.java）

```
mCheatButton = (Button)findViewById(R.id.cheat_button);
mCheatButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Start CheatActivity
        Intent intent = new Intent(QuizActivity.this, CheatActivity.class);
        startActivity(intent);
    }
});
```

在启动activity前，`ActivityManager`会确认指定的`Class`是否已在manifest配置文件中声明。如果已完成声明，则启动activity，应用正常运行。反之，则抛出`ActivityNotFoundException`异常，应用崩溃。这就是必须在manifest配置文件中声明应用的全部activity的原因。

运行GeoQuiz应用。单击CHEAT!按钮，新activity实例的用户界面将显示在屏幕上。点击后退按钮，`CheatActivity`实例会被销毁，`QuizActivity`实例的用户界面又出现了。

显式intent与隐式intent

如果通过指定`Context`与`Class`对象，然后调用intent的构造方法来创建`Intent`，则创建的是显式intent。在同一应用中，我们使用显式intent来启动activity。

同一应用里的两个activity，却要借助于应用外部的`ActivityManager`通信，这似乎有点怪。不过，这种模式会让不同应用间的activity交互变得容易很多。

一个应用的activity如需启动另一个应用的activity，可通过创建隐式intent来处理。我们会在第15章学习使用隐式intent。

5.3　activity 间的数据传递

QuizActivity和CheatActivity都已就绪，现在可以考虑它们之间的数据传递了。图5-9展示了两个activity间传递的数据信息。

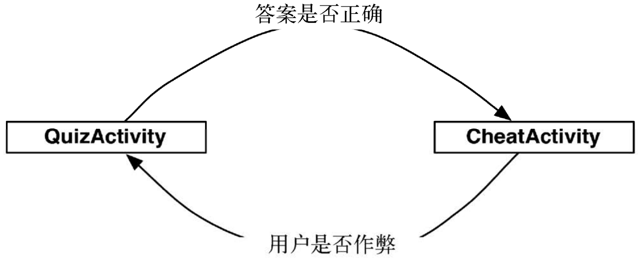


图5-9　QuizActivity与CheatActivity的对话

CheatActivity启动后，QuizActivity会通知它当前问题的答案。

用户知道答案后，点击后退键回到QuizActivity，CheatActivity随即被销毁。在销毁前的瞬间，它会将用户是否作弊的数据传递给QuizActivity。

接下来，首先处理从QuizActivity到CheatActivity的数据传递。

5.3.1　使用 intent extra

为通知CheatActivity当前问题的答案，需将以下语句的返回值传递给它：

```
mQuestionBank[mCurrentIndex].isAnswerTrue()
```

该值将作为extra信息，附加在传入startActivity(Intent)方法的Intent上发送出去。

extra信息可以是任意数据，它包含在Intent中，由启动方activity发送出去。可以把extra信息想象成构造函数参数，虽然我们无法使用带activity子类的构造函数。（Android创建activity实例，并负责管理其生命周期。）接受方activity接收到操作系统转发的intent后，访问并获取其中的extra数据信息，如图5-10所示。

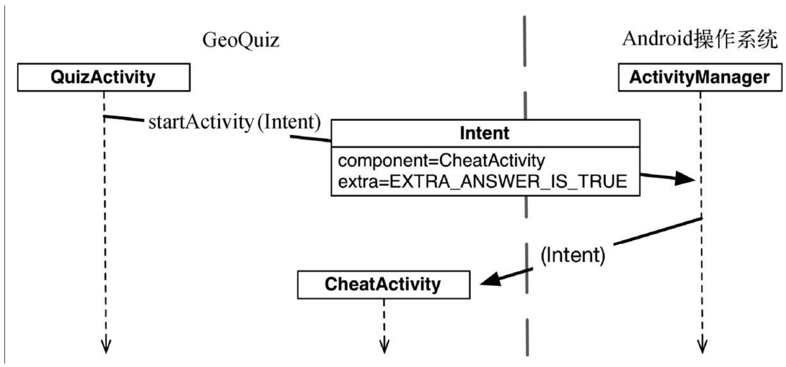


图5-10　intent extra：activity间的通信与数据传递

如同QuizActivity.onSaveInstanceState(Bundle)方法中用来保存mCurrentIndex值的键值结构，extra也是一种键值结构。

要将extra数据信息添加给intent，需要调用Intent.putExtra(...)方法。确切地说，是调用如下方法：

```
public Intent putExtra(String name, boolean value)
```

Intent.putExtra(...)方法形式多变。不变的是，它总是有两个参数。一个参数是固定为String类型的键，另一个参数是键值，可以是多种数据类型。该方法返回intent自身，因此，需要时可进行链式调用。

在CheatActivity.java中，为extra数据信息新增键–值对中的键，如代码清单5-8所示。

代码清单5-8　添加extra常量（CheatActivity.java）

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    ...
```

activity可能启动自不同的地方，所以，应该在获取和使用extra信息的activity那里，为它定义键。如代码清单5-8所示，记得使用包名修饰extra数据信息，这样，可避免来自不同应用的extra间发生命名冲突。

现在，可以返回到QuizActivity，将extra附加到intent上。不过我们有个更好的实现方法。对于CheatActivity处理extra信息的实现细节，QuizActivity和应用的其他代码无需知道。因此，我们可转而在newIntent(...)方法中封装这些逻辑。

在CheatActivity中，创建newIntent(...)方法，如代码清单5-9所示。

代码清单5-9　CheatActivity中的newIntent(...)方法（CheatActivity.java）

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
            "com.bignerdranch.android.geoquiz.answer_is_true";

    public static Intent newIntent(Context packageContext, boolean answerIsTrue) {
        Intent intent = new Intent(packageContext, CheatActivity.class);
        intent.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
        return intent;
    }
    ...
```

使用新建的静态方法，可以正确创建Intent，它配置有CheatActivity需要的extra。answerIsTrue布尔值以EXTRA_ANSWER_IS_TRUE常量放入intent以供解析。利用这种方式，配置传递intent是不是容易多了？

在QuizActivity的按钮监听器中，应用newIntent(...)方法，如代码清单5-10所示。

代码清单5-10 用extra启动（QuizActivity.java）

```
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Start CheatActivity
                Intent intent = new Intent(QuizActivity.this, CheatActivity.class);
                boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
                Intent intent = CheatActivity.newIntent(QuizActivity.this, answerIsTrue);
                startActivity(intent);
            }
        });
```


这里只需一个extra，但如果有需要，也可以附加多个extra到同一个Intent上。如果附加多个extra，也要给newIntent(...)方法相应添加多个参数。

要从extra获取数据，会用到如下方法：

```
public boolean getBooleanExtra(String name, boolean defaultValue)
```

第一个参数是extra的名字。getBooleanExtra(...)方法的第二个参数是指定默认值（默认答案），它在无法获得有效键值时使用。

在CheatActivity代码中，编写代码实现从extra获取信息，存入成员变量中，如代码清单5-11所示。

代码清单5-11 获取extra信息（CheatActivity.java）

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
            "com.bignerdranch.android.geoquiz.answer_is_true";

    private boolean mAnswerIsTrue;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);
    }
    ...
}
```

请注意，Activity.getIntent()方法返回了由startActivity(Intent)方法转发的Intent对象。

最后，在CheatActivity代码中，实现单击SHOW ANSWER按钮后获取答案并将其显示在TextView上，如代码清单5-12所示。

代码清单5-12 提供作弊机会（CheatActivity.java）

```
public class CheatActivity extends AppCompatActivity {
```

```
    ...
    private boolean mAnswerIsTrue;

    private TextView mAnswerTextView;
    private Button mShowAnswerButton;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);

        mAnswerTextView = (TextView) findViewById(R.id.answer_text_view);

        mShowAnswerButton= (Button) findViewById(R.id.show_answer_button);
        mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
            }
        });
    }
}
```

以上代码比较直观。TextView.setText(int)方法用来设置TextView要显示的文字。TextView.setText(...)方法有多种变体。这里通过传入资源ID调用该方法。

运行GeoQuiz应用。单击CHEAT!按钮弹出CheatActivity的用户界面，然后单击SHOW ANSWER按钮查看当前问题的答案。

5.3.2　从子 activity 获取返回结果

现在，用户可以毫无顾忌地偷看答案了。如果CheatActivity能把用户是否看过答案的情况通知给QuizActivity就更好了。下面我们来解决这个问题。

需要从子activity获取返回信息时，可调用以下Activity方法：

```
public void startActivityForResult(Intent intent, int requestCode)
```

该方法的第一个参数同前述的intent。第二个参数是请求代码。请求代码是先发送给子activity，然后再返回给父activity的整数值，由用户定义。在一个activity启动多个不同类型的子activity，且需要判断消息回馈方时，就会用到该请求代码。虽然QuizActivity只启动一种类型的子activity，但为应对未来的需求变化，现在就应设置请求代码常量。

在QuizActivity中，修改mCheatButton的监听器，调用startActivityForResult(Intent, int)方法，如代码清单5-13所示。

代码清单5-13 调用startActivityForResult(...)方法（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";
    private static final int REQUEST_CODE_CHEAT = 0;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Start Cheat Activity
                boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
                Intent intent = CheatActivity.newIntent(QuizActivity.this, answerIsTrue);
                startActivity(intent);
                startActivityForResult(intent, REQUEST_CODE_CHEAT);
            }
        });
```


1. 设置返回结果

实现子activity发送返回信息给父activity，有以下两种方法可用：

```
public final void setResult(int resultCode)
public final void setResult(int resultCode, Intent data)
```

一般来说，参数resultCode可以是以下任意一个预定义常量。

- Activity.RESULT_OK
- Activity.RESULT_CANCELED

（如需自己定义结果代码，还可使用另一个常量：RESULT_FIRST_USER。）

在父activity需要依据子activity的完成结果采取不同操作时，设置结果代码就非常有用。

例如，假设子activity有一个OK按钮和一个Cancel按钮，并且每个按钮的单击动作分别设置有不同的结果代码。那么，根据不同的结果代码，父activity就能采取不同的操作。

子activity可以不调用setResult(...)方法。如果不需要区分附加在intent上的结果或其他信息,可让操作系统发送默认的结果代码。如果子activity是以调用startActivityForResult(...)方法启动的，结果代码则总是会返回给父activity。在没有调用setResult(...)方法的情况下，如果用户按了后退按钮，父activity则会收到Activity.RESULT_CANCELED的结果代码。

2. 返还intent

GeoQuiz应用中，数据信息需要回传给QuizActivity。因此，我们需要创建一个Intent，附加上extra信息后，调用Activity.setResult(int, Intent)方法将信息回传给QuizActivity。

在CheatActivity代码中，为extra的键增加常量，再创建一个私有方法，用来创建intent、附加extra并设置结果值。然后在SHOW ANSWER按钮的监听器代码中调用该方法。设置结果值的方法如代码清单5-14所示。

代码清单5-14　设置结果值（CheatActivity.java）

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    private static final String EXTRA_ANSWER_SHOWN =
        "com.bignerdranch.android.geoquiz.answer_shown";
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
                setAnswerShownResult(true);
            }
        });
    }

    private void setAnswerShownResult(boolean isAnswerShown) {
        Intent data = new Intent();
        data.putExtra(EXTRA_ANSWER_SHOWN, isAnswerShown);
        setResult(RESULT_OK, data);
    }

}
```

用户单击SHOW ANSWER按钮时，`CheatActivity`调用`setResult(int, Intent)`方法将结果代码以及intent打包。

然后，在用户按后退键回到`QuizActivity`时，`ActivityManager`调用父activity的以下方法：

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
```

该方法的参数来自`QuizActivity`的原始请求代码以及传入`setResult(int, Intent)`方法的结果代码和intent。

图5-11展示了应用内部的交互时序。

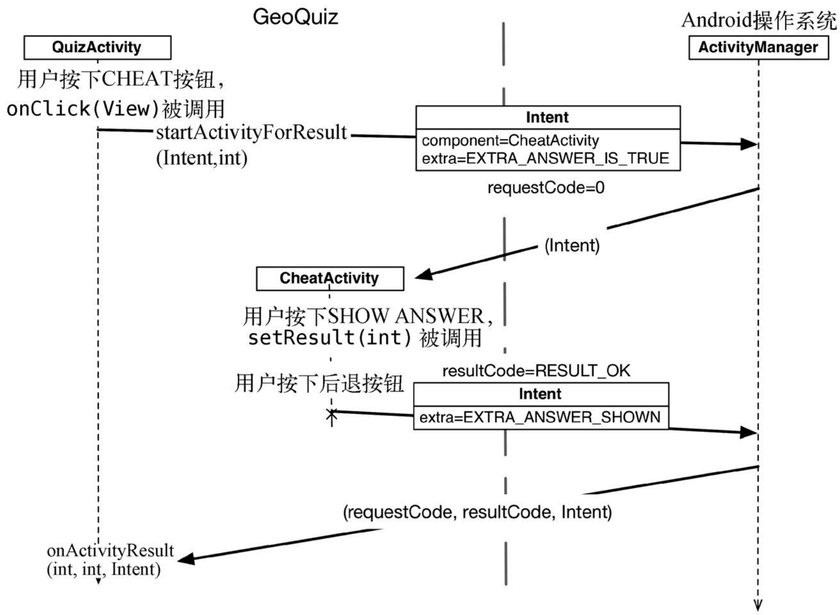


图5-11 GeoQuiz应用内部的交互时序图

最后覆盖`QuizActivity`的`onActivityResult(int, int, Intent)`方法来处理返回结果。然而，结果intent的内容也是`CheatActivity`的实现细节，因而还要添加另一个方法协助解析出`QuizActivity`能用的信息，如代码清单5-15所示。

代码清单5-15 解析结果intent（CheatActivity.java）

```
public static Intent newIntent(Context packageContext, boolean answerIsTrue) {
    Intent intent = new Intent(packageContext, CheatActivity.class);
    intent.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
    return intent;
}

public static boolean wasAnswerShown(Intent result) {
    return result.getBooleanExtra(EXTRA_ANSWER_SHOWN, false);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
}
```

3. 处理返回结果

在QuizActivity.java中，新增一个成员变量保存`CheatActivity`回传的值。然后覆盖`onActivityResult(...)`方法获取它。别忘了检查请求代码和返回代码是否符合预期。实践证明，这样做会方便将来的代码维护。`onActivityResult(...)`方法的实现如代码清单5-16所示。

代码清单5-16　onActivityResult(...)方法的实现（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {
    ...
    private int mCurrentIndex = 0;
    private boolean mIsCheater;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }

        if (requestCode == REQUEST_CODE_CHEAT) {
            if (data == null) {
                return;
            }
            mIsCheater = CheatActivity.wasAnswerShown(data);
        }
    }
    ...
}
```

最后，修改QuizActivity中的checkAnswer(boolean)方法，确认用户是否偷看答案并作出相应的反应。基于mIsCheater变量值改变toast消息的做法如代码清单5-17所示。

代码清单5-17　基于mIsCheater变量值改变toast消息（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            mIsCheater = false;
            updateQuestion();
        }
    });
    ...
}
...
private void checkAnswer(boolean userPressedTrue) {
    boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();

    int messageResId = 0;

    if (mIsCheater) {
```

```
            messageResId = R.string.judgment_toast;
        } else {
            if (userPressedTrue == answerIsTrue) {
                messageResId = R.string.correct_toast;
            } else {
                messageResId = R.string.incorrect_toast;
            }
        }

        Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
            .show();
    }
```

运行GeoQuiz应用。偷看一下答案，看看会发生什么。

5.4 activity 的使用与管理

来看看当我们在各activity间往返的时候，操作系统层面到底发生了什么。首先，在桌面启动器中点击GeoQuiz应用时，操作系统并没有启动应用，而只是启动了应用中的一个activity。确切地说，它启动了应用的launcher activity。在GeoQuiz应用中，`QuizActivity`就是它的launcher activity。

使用应用向导创建GeoQuiz应用以及`QuizActivity`时，`QuizActivity`默认被设置为launcher activity。配置文件中，`QuizActivity`声明的`intent-filter`元素节点下，可看到`QuizActivity`被指定为launcher activity，如代码清单5-18所示。

代码清单5-18 `QuizActivity`被指定为launcher activity（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ... >

    <application
        ... >

        <activity android:name=".QuizActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".CheatActivity" />
        </activity>
    </application>

</manifest>
```

`QuizActivity`实例出现在屏幕上后，用户可单击CHEAT!按钮。`CheatActivity`实例随即在`QuizActivity`实例上被启动。此时，它们都处于activity栈中，如图5-12所示。

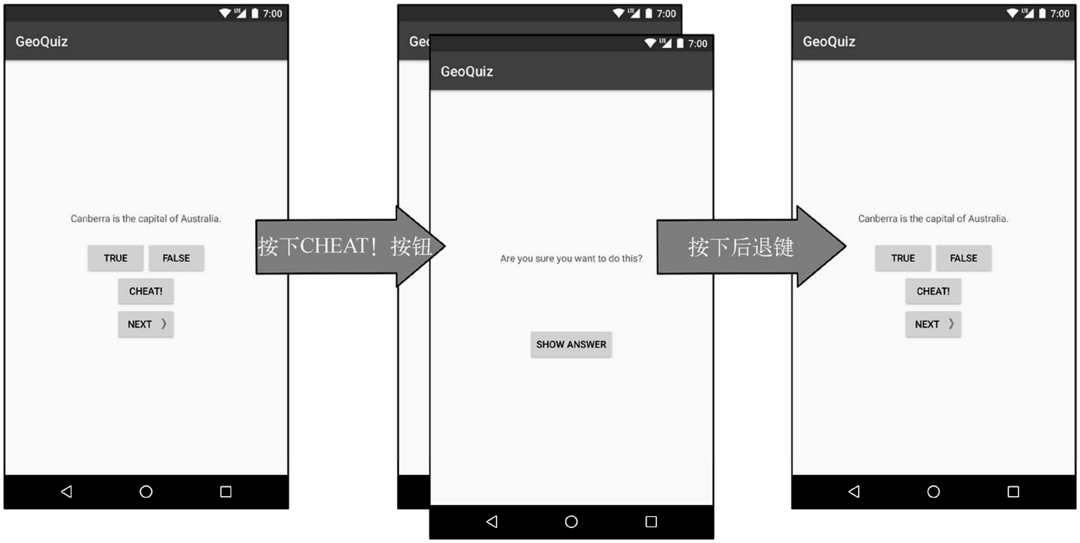


图5-12　GeoQuiz的回退栈

按后退键，`CheatActivity`实例被弹出栈外，`QuizActivity`重新回到栈顶部，如图5-12所示。

在`CheatActivity`中调用`Activity.finish()`方法同样可以将`CheatActivity`从栈里弹出。

如果运行GeoQuiz应用，在`QuizActivity`界面按后退键，`QuizActivity`将从栈里弹出，我们将退回到GeoQuiz应用运行前的画面，如图5-13所示。

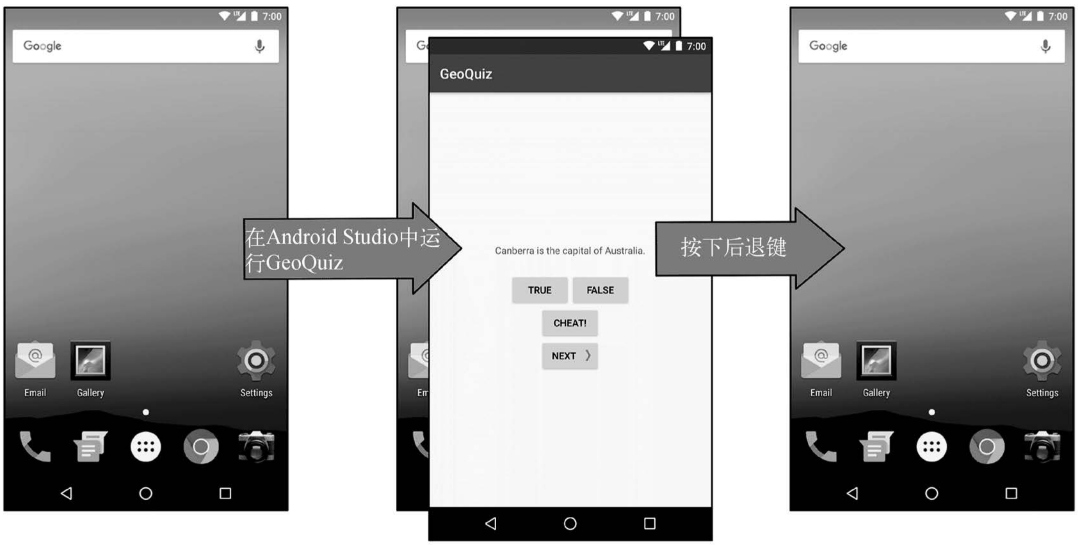


图5-13　后退返回至桌面

如果从桌面启动器启动GeoQuiz应用，在QuizActivity界面按后退键，将退回到桌面启动器界面，如图5-14所示。

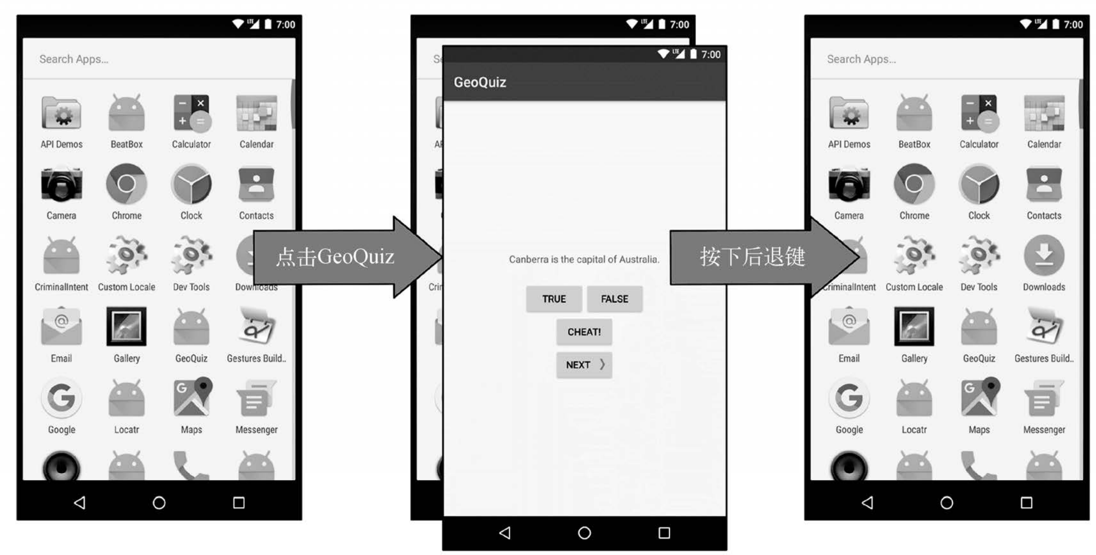


图5-14　从桌面启动器启动GeoQuiz应用

在桌面启动器界面，按后退键，将返回到桌面启动器启动前的系统界面。

至此，可以看到，ActivityManager维护着一个非特定应用独享的*回退栈*。所有应用的activity都共享该回退栈。这也是将ActivityManager设计成操作系统级的activity管理器来负责启动应用activity的原因之一。显然，回退栈是作为一个整体共享于操作系统及设备，而不单单用于某个应用。

（想了解“向上”按钮？我们将在第13章学习如何使用并配置它。）

5.5　挑战练习：堵住漏洞

作弊者是注定会失败的。当然，如果他们能一直避开反作弊手段，那就另当别论了。正所谓道高一尺，魔高一丈，也许他们能做到。

GeoQuiz应用有些大漏洞，你的任务就是堵住它们。从易到难，以下为待解决的三个漏洞。

- 用户作弊后，可以旋转CheatActivity来清除作弊痕迹。
- 作弊返回后，用户可以旋转QuizActivity来清除mIsCheater变量值。
- 用户可以不断单击NEXT按钮，跳到偷看过答案的问题，从而使作弊纪录丢失。

祝好运！

第6章 Android SDK版本与兼容

开发完GeoQuiz应用，你已经有了初步的开发体会。本章中，我们学习Android系统版本的相关知识。在学习本书后续章节，以及应对未来实际的复杂应用开发时，你就会明白掌握本章内容有多么重要。

6.1 Android SDK 版本

表6-1显示了各SDK版本、相应的Android固件版本，以及截至2016年12月使用各版本的设备比例。

表6-1　Android API级别、固件版本以及在用设备比例

API级别	代　　号	设备固件版本	在用设备比例（%）
24	Nougat	7.0	0.4
23	Marshmallow	6.0	26.3
22	Lollipop	5.1	23.2
21		5.0	10.8
19	KitKat	4.4	24.0
18	Jelly Bean	4.3	1.9
17		4.2	6.4
16		4.1	4.5
15	Ice Cream Sandwich（ICS）	4.0.3，4.0.4	1.2
10	Gingerbread	2.3.3 – 2.3.7	1.2
8	Froyo	2.2	0.1

* 注意，本表已忽略比例低于0.1%的在用设备。

每一个有发布代号的版本随后都会有相应的增量版本。例如，Ice Cream Sandwich最初的发布版本为Android 4.0（API 14级），但没过多久，Android 4.0.3及4.0.4（API 15级）的增量发行版本就取代了它。

当然，表6-1中的比例会动态变化，但这些数字已揭示一种重要趋势，即新版本发布后，运行老版本的Android设备是不会立即进行升级或更换的。截至2016年12月，超过15%的设备仍然运行着Jelly Bean或更早版本的系统。Android 4.3（Jelly Bean最后的升级版本）发布于2013年10月。

（感兴趣的话，可去http://developer.android.com/about/dashboards/index.html查看表6-1数据的动态更新。）

为什么仍有这么多设备运行着老版本Android系统？主要原因在于Android设备生产商和运营商之间的激烈竞争。每个运营商都希望拥有专属定制机。设备生产商也有同样的压力——所有手机都基于相同的操作系统，而他们又想与众不同。最终，屈服于市场和运营商的双重压力，各种专属的、无法升级的定制版Android设备涌向市场，令人眼花缭乱、目不暇接。

定制版Android设备不能运行Google发布的新版本Android系统。因此，用户只能寄望于定制版的兼容升级。然而，即便可以获得这种升级，通常也是Google新版本发布后数月的事情了。生产商往往更愿意投入资源推出新设备，而不是持续升级旧设备。

6.2　Android 编程与兼容性问题

各种设备迟缓的版本升级再加上Google定期的新版本发布，给Android编程带来了严重的兼容性问题。为扩大市场份额，对于运行KitKat、Lollipop、Marshmallow、Nougat以及任何最新版本的Android设备（还要考虑各种尺寸），Android开发人员必须保证应用都能兼容。

还好，开发应用时，不同尺寸设备的处理要比想象中的简单。手机屏幕尺寸虽然繁多，但Android布局系统为编程适配做了很好的工作。平板设备处理起来会复杂一些，但配置修饰符可用来处理屏幕适配（详见第17章）。不过，对于同样运行着Android系统的Android TV和Android Wear设备，由于UI差异太大，应用的交互模式和设计通常需要重新考虑。

6.2.1　比较合理的版本

本书支持的最老版本是API 19级（KitKat）。虽然还在支持，但我们更应该将精力投入在较新系统版本上（API 19+级）。当前，Gingerbread、Ice Cream Sandwich和Jelly Bean系统版本的市场份额正逐月下降，还在这些老设备上投入过多显然得不偿失。

对于增量版本，向下兼容一般问题不大。主要版本向下兼容才是大麻烦。也就是说，仅支持5.x版本的工作量不大，但需要支持到4.x的话，考虑到这么多不同版本的差异，工作量就相当大了。谢天谢地，Google提供了一些兼容库，大大降低了开发难度。后续章节会介绍具体内容。

Honeycomb版本的发布是Android世界的一个分水岭，该版本引入了全新的UI和构造组件。Honeycomb起初只面向平板设备开发，所以直到发布Ice Cream Sandwich，这些新功能才正式发布给终端用户使用。随后它又经历了几次增量版本升级。

尽管Android以及第三方库提供了大量的兼容性编程支持，但兼容性问题已实实在在地增加了学习的复杂性。

新建GeoQuiz项目时，在新建应用向导界面，我们设置过最低SDK版本，如图6-1所示。（注意，Android的“SDK版本”和“API级别”二词可以交替使用。）

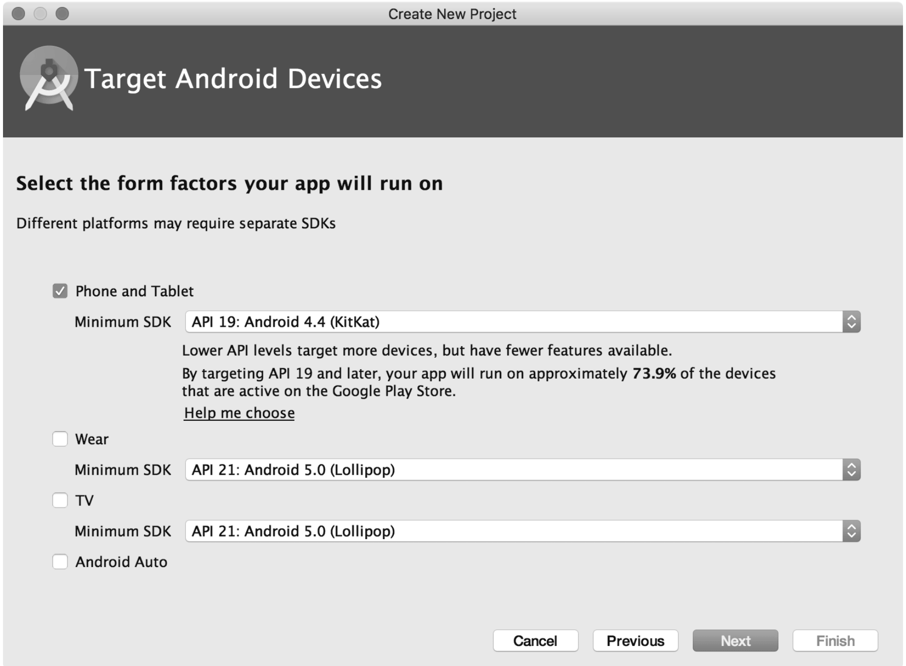


图6-1　创建新项目向导，还有印象吗

除了最低支持版本，还可以设置目标版本和编译版本。下面来看看都有哪些默认选项，以及新建项目时该如何选择。

所有的设置都保存在应用模块的build.gradle文件中。编译版本独占该文件。虽然最低版本和目标版本也设置在该文件中，但它们的作用是覆盖和设置配置文件AndroidManifest.xml。

打开应用模块下的build.gradle文件，查看`compileSdkVersion`、`minSdkVersion`和`targetSdkVersion`的属性值，如代码清单6-1所示。

代码清单6-1　查看编译配置（app/build.gradle）

```
compileSdkVersion 25
buildToolsVersion "25.0.0"

defaultConfig {
    applicationId "com.bignerdranch.android.geoquiz"
    minSdkVersion 19
    targetSdkVersion 25
    ...
}
```

6.2.2　SDK 最低版本

以最低版本设置值为标准，操作系统会拒绝将应用安装在系统版本低于标准的设备上。

例如，设置版本为API 19级（KitKat），便赋予了系统在运行 KitKat 及以上版本的设备上安装GeoQuiz应用的权限。而在运行诸如Jelly Bean版本的设备上，系统会拒绝安装GeoQuiz应用。

再看表6-1，你就会明白为什么选 KitKat 作为SDK最低版本比较合适，因为有超过80%的在用设备支持安装此应用。

6.2.3 SDK 目标版本

目标版本的设定值告知Android：应用是为哪个API级别设计的。大多数情况下，目标版本即最新发布的Android版本。

什么时候需要降低SDK目标版本呢？新发布的SDK版本会改变应用在设备上的显示方式，甚至连操作系统后台运行行为都会受影响。如果应用已开发完成，应确认它在新版本上能否如预期那样正常运行。查看http://developer.android.com/reference/android/os/Build.VERSION_CODES.html上的文档，检查可能出现问题的地方。根据分析结果，要么修改应用以适应新版本系统，要么降低SDK目标版本。

降低SDK目标版本可以保证的是，即便在高于目标版本的设备上，应用仍然可以正常运行，且运行行为仍和目标版本保持一致。这是因为新发布版本中的变化已被忽略。

6.2.4 SDK 编译版本

代码清单6-1中，第1行标为`compileSdkVersion`的是SDK编译版本设置。该设置不会出现在manifest配置文件里。SDK最低版本和目标版本会通知给操作系统，而SDK编译版本只是你和编译器之间的私有信息。

Android的特色功能是通过SDK中的类和方法展现的。在编译代码时，SDK编译版本（即编译目标）指定具体要使用的系统版本。Android Studio在寻找类包导入语句中的类和方法时，编译目标确定具体的基准系统版本。

编译目标的最佳选择为最新的API级别（当前级别为25，代号为Nougat）。当然，需要的话，也可以改变应用的编译目标。例如，Android新版本发布时，可能就需要更新编译目标，以便使用新版本引入的方法和类。

可以修改build.gradle文件中的SDK最低版本、目标版本以及编译版本。修改完毕，项目和Gradle更改重新同步后才能生效。选择Tools → Android → Sync Project with Gradle Files菜单项，项目随即会重新编译。

6.2.5 安全添加新版本 API 中的代码

GeoQuiz应用的SDK最低版本和编译版本间的差异较大，由此带来的兼容性问题需要处理。例如，在GeoQuiz应用中，如果调用了KitKat（API 19级）以后的SDK版本中的代码会怎么样呢？结果显示，在 KitKat 设备上安装运行时，应用会崩溃。

这个问题可以说是曾经的测试噩梦。然而，受益于Android Lint的不断改进，现在在老版本

系统上调用新版本代码时，潜在问题在编译时就能被发现。也就是说，如果使用了高版本系统API中的代码，Android Lint会提示编译错误。

目前，GeoQuiz应用中的简单代码都来自于API 19级或更早版本。现在，我们来增加API 21级（Lollipop）的代码，看看会发生什么。

打开CheatActivity.java文件，在SHOW ANSWER按钮的`OnClickListener`方法中，添加代码清单6-2所示代码，从而在隐藏按钮的同时，显示一段圆球特效动画。

代码清单6-2　添加动画特效代码（CheatActivity.java）

```
mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
        setAnswerShownResult(true);

        int cx = mShowAnswerButton.getWidth() / 2;
        int cy = mShowAnswerButton.getHeight() / 2;
        float radius = mShowAnswerButton.getWidth();
        Animator anim = ViewAnimationUtils
                .createCircularReveal(mShowAnswerButton, cx, cy, radius, 0);
        anim.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
                mShowAnswerButton.setVisibility(View.INVISIBLE);
            }
        });
        anim.start();
    }
});
```

传入一些特定参数，`createCircularReveal`方法创建了一个`Animator`。首先，指定要显示或隐藏的`View`，然后是动画的中心位置、起始半径和结束半径。为隐藏按钮，这里将它的起始半径设置为按钮宽度，结束半径设置为0。

动画启动前，设置一个监听器，确定动画何时播完。动画一结束，就显示答案并隐藏按钮。

最后播放动画，看看效果。有关Android动画方面的知识将在第32章学习。

Android直到SDK API 21级才加入`ViewAnimationUtils`类和它的`createCircularReveal`方法。因此，上述代码在低版本设备上运行时会出问题。

输入代码清单6-2所示代码时，Android Lint会立即提示，这段代码对于最低版本SDK是不安全的。如果没看到提示，请选择Analyze → Inspect Code....菜单项手动触发Lint。因为SDK编译版本为API 21级，编译器本身编译代码没有问题，而Android Lint知道项目SDK最低版本，所以及时指出了问题。

虽然Lint提示了类似Call requires API level 21 (Current min is 19)的警告信息，但是你可以忽略它。不过，出了问题可别怪Lint没有提醒你。

该怎么消除这些错误信息呢？一种办法是提升SDK最低版本到21。然而，提升SDK最低版本只是回避了兼容性问题。如果应用不能安装在API 19级和更老版本设备上，那么也就不存在新老系统的兼容性问题了。因此，实际上这并没有真正解决兼容性问题。

比较好的做法是将高API级别代码置于检查Android设备版本的条件语句中，如代码清单6-3所示。

代码清单6-3 首先检查设备的编译版本

```
mShowAnswerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
        setAnswerShownResult(true);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            int cx = mShowAnswerButton.getWidth() / 2;
            int cy = mShowAnswerButton.getHeight() / 2;
            float radius = mShowAnswerButton.getWidth();
            Animator anim = ViewAnimationUtils
                    .createCircularReveal(mShowAnswerButton, cx, cy, radius, 0);
            anim.addListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    super.onAnimationEnd(animation);
                    mShowAnswerButton.setVisibility(View.INVISIBLE);
                }
            });
            anim.start();
        } else {
            mShowAnswerButton.setVisibility(View.INVISIBLE);
        }
    }
});
```

`Build.VERSION.SDK_INT`常量代表了Android设备的版本号。可将该常量同代表Lollipop版本的常量进行比较。（版本号清单可参看网页http://developer.android.com/reference/android/os/Build.VERSION_CODES.html。）

现在动画特效代码只有在API 21级或更高版本的设备上运行应用才会被调用。应用代码在API 19级设备上终于安全了，Android Lint应该也满意了吧。

在Lollipop或更高版本的设备上运行GeoQuiz应用。启动`CheatActivity`时，确认看到了想要的动画特效。

也可以在KitKat设备（虚拟或实体）上运行GeoQuiz应用。当然，动画特效是看不到了，但可验证应用仍能正常运行。

6.3 使用 Android 开发者文档

从Android Lint错误信息中可看到不兼容代码所属的API级别。也可在Android开发者文档里查看各API级别特有的类和方法。

越早熟悉使用开发者文档越有利于开发。没人能记住Android SDK中的海量信息，更不要说定期发布的新版本系统了。因此，学会查阅SDK文档，不断学习新的知识尤显重要。

Android开发者文档是优秀而丰富的信息来源。文档主页是http://developer.android.com/。文档分为三大部分，即设计、开发和发布。设计部分的文档包括应用UI设计的模式和原则。开发部分包括SDK文档和培训资料。发布部分讲述如何在Google Play商店里或以开放发布模式准备并发布应用。有机会的话，一定要仔细研读这些资料。

开发部分又细分为七大块内容。

- ❑ Android培训：初级和中级开发者的培训模块，包括可下载的示例代码。
- ❑ API使用指南：基于主题的应用组件、特色功能详述以及它们的最佳实践。
- ❑ 参考文档：SDK中类、方法、接口、属性常量等可搜索、交叉链接的参考文档。
- ❑ 示例代码：如何使用各种API的示例代码。
- ❑ Android Studio：与Android Studio IDE相关的内容。
- ❑ Android NDK：有关Android原生开发工具的介绍和参考链接，该工具允许开发人员使用C或C++开发应用。
- ❑ Google服务：Google专属API的相关信息，包括Google地图和Google云消息。

Android文档可脱机查看。浏览SDK安装文件所在目录，找到docs目录。该目录包含了全部的Android开发者文档内容。

例如，为确定`ViewAnimationUtils`类所属的API级别，使用文档浏览器右上角的搜索框搜索它。搜索结果显示有多种类别的信息。你想要的结果位于参考文档部分（注意灵活使用左边的搜索过滤功能）。

选择第一条结果，进入`ViewAnimationUtils`类的参考文档页面，如图6-2所示。该页面顶部的链接可以链接到不同的部分。

向下滚动，找到并点击`createCircularReveal(...)`方法查看具体的方法描述。从该方法名的右边可以看到，`createCircularReveal(...)`方法最早被引入的API级别是API 21级。

如果想查看`ViewAnimationUtils`类的哪些方法可用于API 19级，可按API级别过滤引用。在页面左边按包索引的类列表上方，找到API级别过滤框，目前它显示为API level: 21。展开下拉表单，选择数字19。一般而言，所有API 19级以后引入的方法都会被过滤掉，并自动变为灰色。`ViewAnimationUtils`类是在API 21级引入的，所以，可以看到一条该类无法用于API 19级的警示信息。

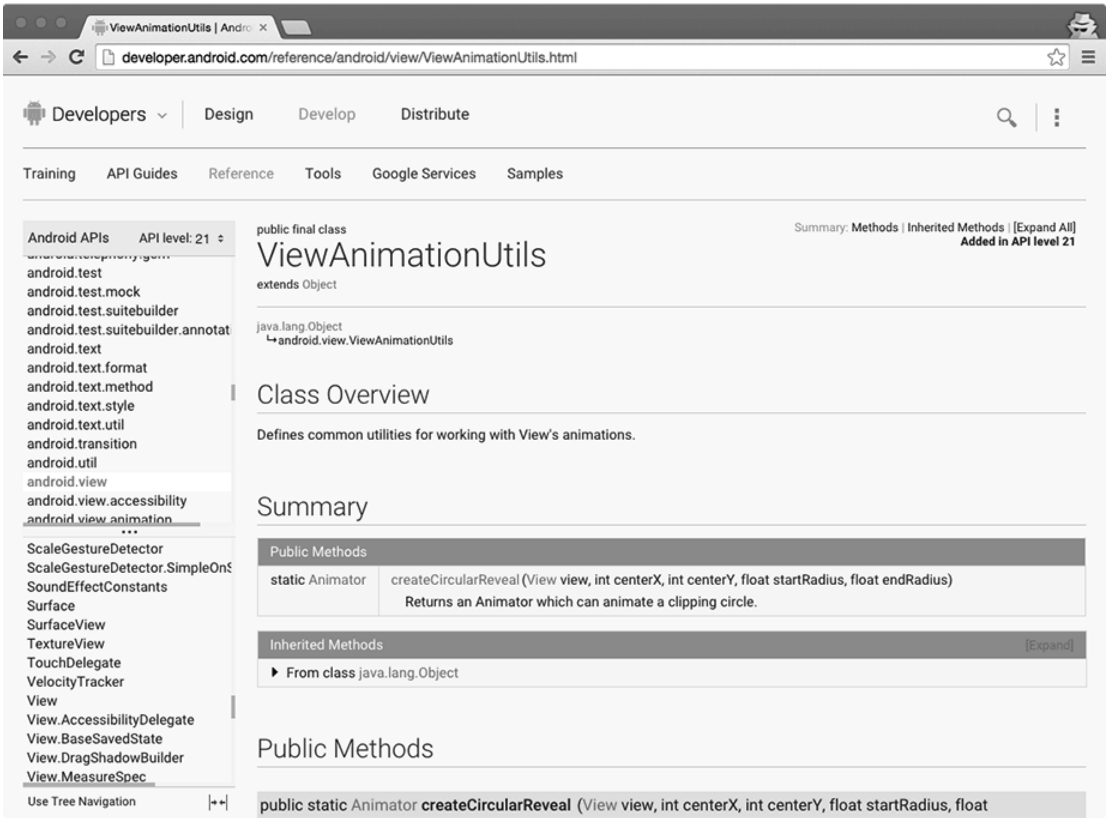


图6-2　ViewAnimationUtils参考文档页面

API级别过滤非常有用，可以让你知道应用要用到的类在哪个API级别可用。例如，在参考文档页搜索`Activity`类，以API 19级过滤。结果显示，诸如`onEnterAnimationComplete`的很多方法是在API 19级才开始添加的。而在Lollipop SDK中，`onEnterAnimationComplete`属于附加方法，为activity间的跳转提供了有趣的过场动画效果。

在后续章节的学习过程中，一定要经常查阅开发者文档。完成章末的挑战练习，以及探究某些类、方法或其他主题时，都需要查阅相关的文档资料。Google还在不断地更新和改进Android文档，新知识和新概念也因此不断涌现，学无止境啊。

6.4　挑战练习：报告编译版本

在GeoQuiz应用的页面布局上添加一个`TextView`组件，向用户报告设备运行系统的API级别，如图6-3所示。

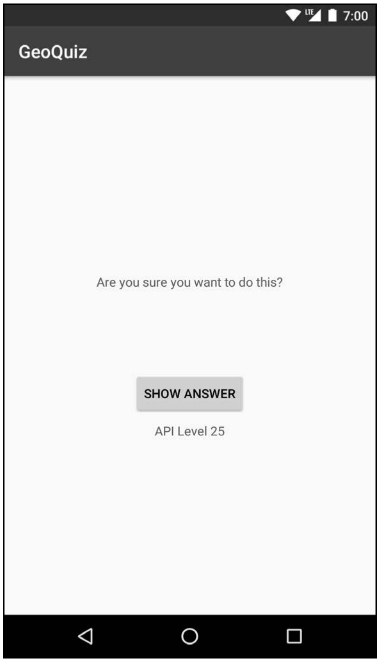


图6-3　完成后的用户界面

应用运行时才能知道设备的编译版本，所以不能直接在布局上设置TextView值。打开Android文档中的TextView参考文档页，查找TextView的文本赋值方法。寻找可以接受字符串或CharSequence的单参数方法。

另外，可使用TextView参考文档里列出的其他XML属性来调整文本的尺寸或样式。

6.5　挑战练习：限制作弊次数

允许用户最多作弊3次。记录用户偷看答案的次数，在CHEAT按钮下显示剩余次数。超出后，禁用偷看按钮。

第 7 章

UI fragment与fragment管理器

本章，我们开始开发一个名为CriminalIntent的应用。该应用可详细记录各种办公室陋习，如随手将脏盘子丢在休息室水池里，或者自己打印完文件就走，全然不顾公共打印机里已缺纸等。

CriminalIntent应用记载的陋习记录包括标题、日期和照片,支持在联系人中查找当事人，通过E-mail、Twitter、Facebook或其他应用提出抗议。看见陋习，记录下来，舒缓了心情，就可以继续专心做手头上的工作。真是个不错的应用。

CriminalIntent应用比较复杂，需要13章的篇幅来完成。应用的用户界面由列表以及记录明细组成。主屏幕会显示已记录陋习的列表清单。用户可新增记录或查看和编辑现有记录，如图7-1所示。

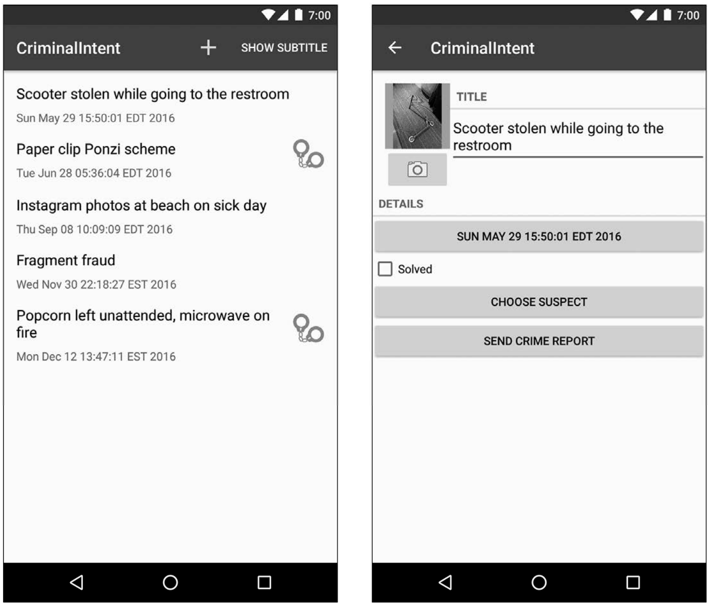


图7-1　CriminalIntent，一个列表明细类应用

7.1　UI 设计的灵活性需求

你可能认为，开发一个由两个activity组成的列表明细类应用就行了，一个activity负责管理记录列表界面，另一个负责管理记录明细界面。点击列表中某条记录会启动其明细activity实例，按后退键会销毁明细activity并返回到记录列表activity界面。想看下条记录，同样操作。

理论上这想法行得通，但如果需要更复杂的用户界面呈现及跳转，怎么办呢?

- 假设用户正在平板设备上运行CriminalIntent应用。平板以及大尺寸手机的屏幕较大，能够同时显示列表和记录明细（最起码在横屏模式下是这样），如图7-2所示。

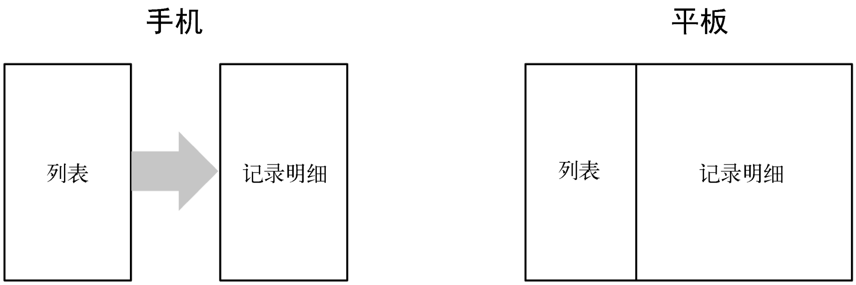


图7-2　手机和平板上理想的列表明细界面

- 假设用户正在手机上查看记录明细信息，并想查看列表中的下一条记录信息。如果无需返回列表界面，滑动屏幕就能查看下一条记录就好了。每滑动一次屏幕，应用便自动切换到下一条记录明细。

可以看出，灵活多变的UI设计是以上假设情景的共同点。也就是说，为了适应用户或设备的需求，activity界面可以在运行时组装，甚至重新组装。

activity自身并不具有这样的灵活性。activity视图可以在运行时切换，但控制视图的代码必须在activity中实现。结果，各个activity还是得和特定的用户界面紧紧绑定。

7.2　引入 fragment

采用fragment而不是activity来管理应用UI，可绕开Android系统activity使用规则的限制。

fragment是一种控制器对象，activity可委派它执行任务。这些任务通常就是管理用户界面。受管的用户界面可以是一整屏或是整屏的一部分。

管理用户界面的fragment又称为UI fragment。它自己也有产生于布局文件的视图。fragment视图包含了用户可以交互的可视化UI元素。

activity视图能预留位置供fragment视图插入。本章只需要插入一个fragment。如果有多个fragment要插入，activity视图就提供多个位置。

根据应用和用户的需求，可联合使用fragment及activity来组装或重组用户界面。在整个生命周期过程中，activity视图还是那个视图。因此不必担心会违反Android系统的activity使用规则。

下面来看看应用该如何支持同屏显示列表与明细内容。实际上，这类应用的activity视图由列表fragment和明细fragment组成。明细视图负责显示列表项的明细内容。

选择不同的列表项就显示对应的明细视图，activity负责以一个明细fragment替换另一个明细fragment，如图7-3所示。这样，视图切换的过程中，也不用销毁activity了。有fragment助阵，一切就这么简单。



图7-3　明细fragment的切换

除列表明细类应用外，使用UI fragment将应用的UI分解成构建块，还适用于其他类型的应用。例如，利用单个构建块，可以方便地构建分页界面、动画侧边栏界面等更多定制界面。

你还会在第11章和第17章中体会到fragment的妙处。不过，UI设计得这样灵活也是要付出代价的，即更复杂的应用、更多的部件以及大量的实现代码。现在先来感受它复杂的一面。

7.3　着手开发 CriminalIntent

CriminalIntent应用比较复杂，第一步先开发应用的记录明细部分。完成后的界面如图7-4所示。



图7-4　本章结束时，CriminalIntent应用的界面

首先设计一个名为CrimeFragment的UI fragment来管理图7-4所示的用户界面，再设计一个名为CrimeActivity的activity来托管CrimeFragment实例。

可以这样理解托管：activity在其视图层级里提供一处位置，用来放置fragment视图，如图7-5所示。fragment本身没有在屏幕上显示视图的能力。因此，只有将它的视图放置在activity的视图层级结构中，fragment视图才能显示在屏幕上。

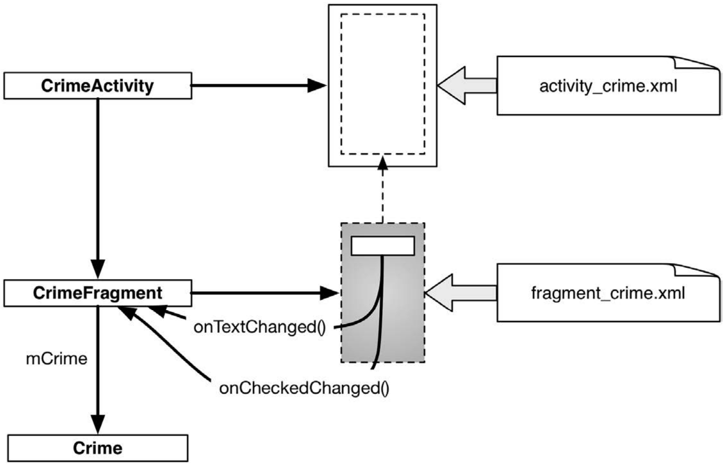


图7-5　CrimeActivity托管着CrimeFragment

CriminalIntent是个大型项目，借助对象图解可以更好地理解它。图7-6展示了CriminalIntent项目涉及的对象以及对象间的关系。可以不去记忆，但开工前，整体了解开发目标将非常有帮助。

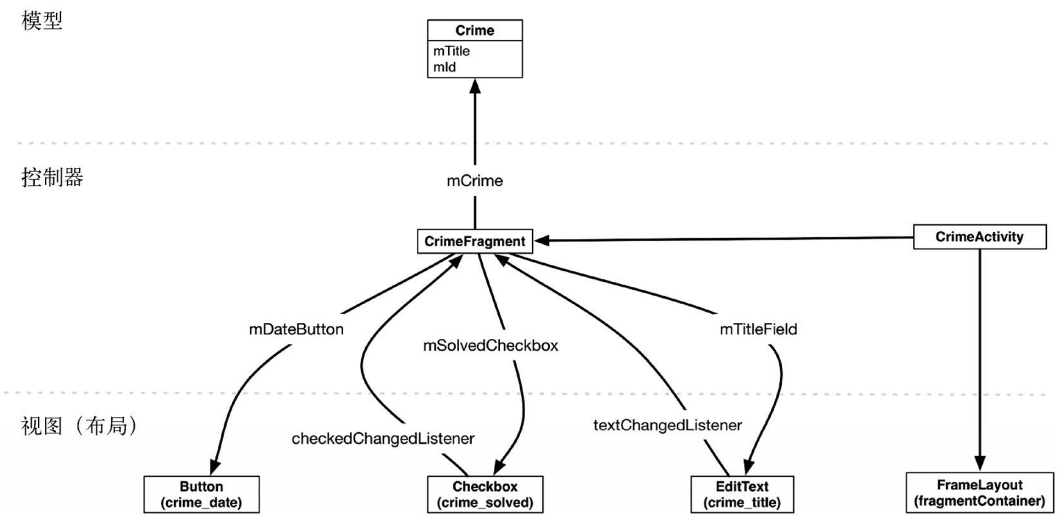


图7-6　CriminalIntent应用的对象图解（本章应完成部分）

可以看到，CrimeFragment的作用与activity在GeoQuiz应用中的作用差不多，都负责创建并管理用户界面，以及与模型对象进行交互。

图7-6中的Crime、CrimeFragment以及CrimeActivity是我们要开发的三个类。

Crime实例代表某种办公室陋习。在本章中，一个crime有一个标题、一个标识ID、一个日期和一个布尔值。布尔值用于表示陋习是否被解决。标题是一段描述性名称，如“向水槽中倾倒有毒物”或“某人偷了我的酸奶!”等。标识ID是识别Crime实例的唯一元素。

简单起见，本章只使用一个Crime实例，并将其存放在CrimeFragment类的成员变量（mCrime）中。

CrimeActivity视图由FrameLayout组件组成，FrameLayout组件为CrimeFragment视图安排了显示位置。

CrimeFragment视图由一个LinearLayout组件及其三个子视图组成。这三个子视图包括一个EditText组件、一个Button组件和一个CheckBox组件。CrimeFragment类中有存储它们的成员变量，并设有监听器，会响应用户操作时，更新模型层数据。

7.3.1　创建新项目

介绍了这么多，是时候创建新应用了。选择File → New → New Project...菜单项创建新的Android应用。如图7-7所示，将应用命名为CriminalIntent，并在Company Domain一栏填入android.bignerdranch.com。

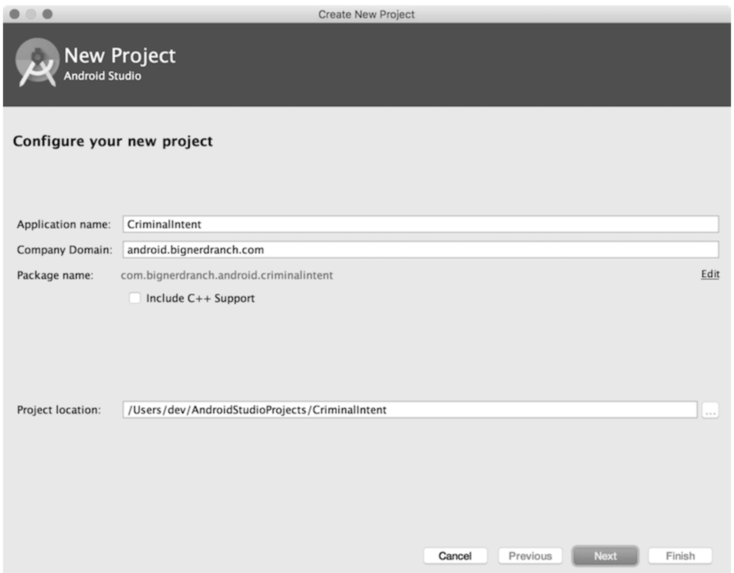


图7-7　创建CriminalIntent应用

单击Next按钮，指定SDK最低版本为API 19: Android 4.4，并确认仅勾选了Phone and Tablet

设备类型。

单击Next按钮，选择新增Empty Activity，单击Next继续。

最后，命名activity为`CrimeActivity`，单击Finish按钮完成，如图7-8所示。

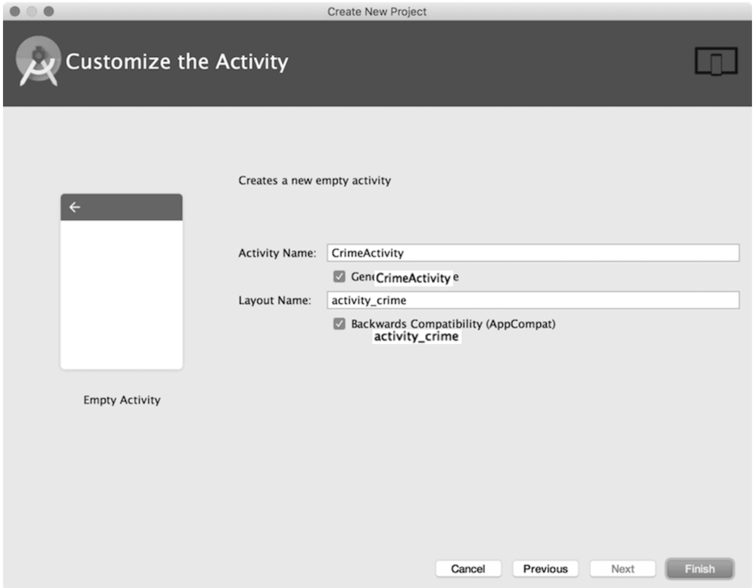


图7-8　创建`CrimeActivity`

7.3.2　两类 fragment

fragment是在API 11级系统版本中引入的，当时Google发布了第一台平板设备。可以说，UI设计要灵活，就是为了平板这样的大屏幕设备。现在，Google有两个版本的fragment实现可供选择：原生版本和支持库版本。

原生版本的fragment实现内置在设备系统中。如果应用要支持各个系统版本，在不同设备上运行的fragment可能会有不同的表现。（这主要是因为各个版本的维护有差异，例如，某个bug在4.4系统版本里已修正，而在4.0里却没有。）支持库版本的fragment在类库里，发布时，会打包在应用里。显然，使用支持库fragment的应用，无论在哪台设备上运行，都会有相同的表现。

CriminalIntent应用选择使用支持库中的fragment实现。考虑到fragment API不断引入新特性以及支持库不断更新的现状，这一选择还是比较明智的（进一步解释，请参见7.9节）。

7.3.3　在 Android Studio 中增加依赖关系

我们要使用的支持库版fragment来自于AppCompat库。Google为开发者提供了很多兼容库，这是其中一个。本书至始至终都会使用它。欲进一步了解这个库，请参见第13章。

要使用AppCompat支持库，项目必须将其列入依赖关系。打开应用模块下的build.gradle文件。每个项目都有两个build.gradle文件。一个用于整个项目，另一个用于应用模块。我们要编辑的是app/build.gradle文件。

代码清单7-1　Gradle依赖设置（app/build.gradle）

```
apply plugin: 'com.android.application'

android {
    ...
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    ...
    compile 'com.android.support:appcompat-v7:25.0.1'
    ...
}
```


build.gradle待编辑文件的当前依赖设置类似于代码清单7-1。也就是说，项目依赖于libs目录下的所有jar包。Android Studio在创建项目时，也会自动引入其他依赖库。AppCompat库很可能就已经包括在内了。

Gradle允许设置未复制到项目中的依赖项。应用编译时，Gradle会找到并下载依赖包，并将其自动导入到项目中。我们要做的就是预先写好指令，剩下的就交给Gradle了。

如果没有发现AppCompat库，可使用Android Studio提供的工具添加。选择File → Project Structure...菜单项打开项目结构对话框。

选择左边的应用模板，然后在右边点击Dependencies选项页。可以看到，应用模板的依赖项都列在这里了，如图7-9所示。

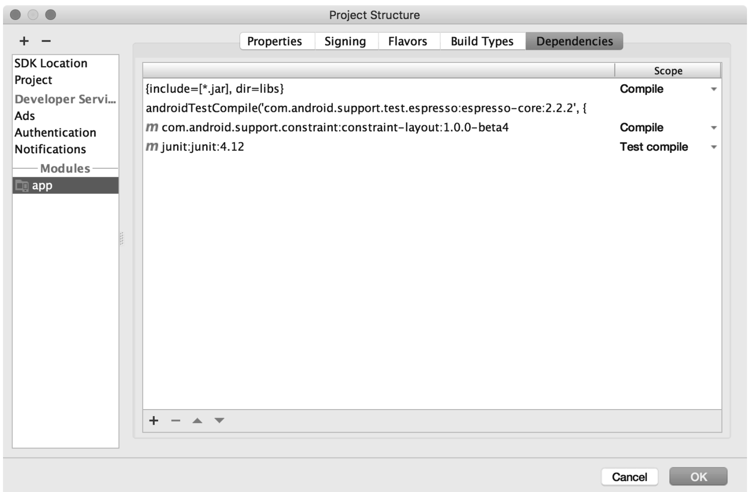


图7-9　app依赖项

AppCompat依赖项可能已列在其中。如果没有，单击+按钮，选择Library dependency界面添加新的依赖项，如图7-10所示。从列表中选中appcompat-v7库后单击OK按钮确认。

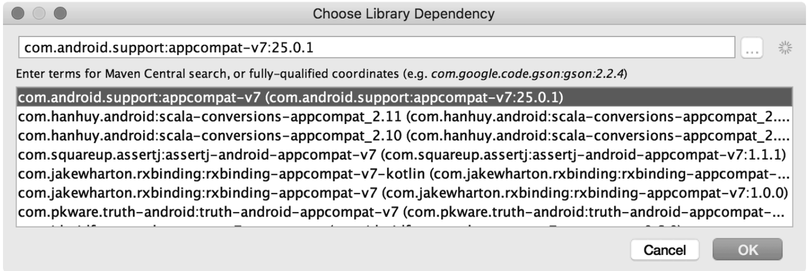


图7-10　可选依赖项

回到app/build.gradle文件的编辑窗口，AppCompat依赖项已添加完毕，如代码清单7-1所示。（如果手动修改了app/build.gradle文件，就需要同步项目和Gradle文件以更新修改内容。同步就是要求Gradle通过下载或删除依赖项，基于修改重新编译。选择Tools → Android → Sync Project with Gradle Files菜单项可手动执行同步。）

代码清单7-1中，灰底部分的依赖项字符串使用了Maven坐标模式：`groupId:artifactId:version`。（Maven是一个依赖包管理工具，详见其官方网站https://maven.apache.org/。）

`groupId`通常是类库的基础包名，能唯一标识Maven仓库中的依赖类库，如上例中的com.android.support。

`artifactId`是包中的特定库名，我们指定的是appcompat-v7。

最后一项`version`是指类库的版本号。CriminalIntent应用依赖于版本号为25.0.1的appcompat-v7库。这是本书截稿时的最新版本。当然，我们的项目支持这之后的任一新版本。为了使用更新的API以及受益于bug修复，建议使用最新版本的支持库。假如Android Studio更新了新的支持库，就不要回退了，使用最新版本就好。

项目依赖的支持库已设置好了，可以用了。在包浏览器中，找到并打开CrimeActivity.java文件。将`CrimeActivity`的超类改为`AppCompatActivity`，如代码清单7-2所示。

代码清单7-2　修改模板代码（CrimeActivity.java）

```
public class CrimeActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);
    }

}
```

进一步完善`CrimeActivity`类之前，我们先来为CriminalIntent应用创建模型层的`Crime`类。

7.3.4 创建 Crime 类

在项目工具窗口中，右键单击com.bignerdranch.android.criminalintent包，选择New → Java Class菜单项。在新建类对话框中，命名类为Crime，单击OK按钮完成。

在随后打开的Crime.java中，增加代码清单7-3所示的代码。

代码清单7-3 Crime类的新增代码（Crime.java）

```
public class Crime {

    private UUID mId;
    private String mTitle;
    private Date mDate;
    private boolean mSolved;

    public Crime() {
        mId = UUID.randomUUID();
        mDate = new Date();

    }
}
```

UUID是Android框架里的Java工具类。有了它，产生唯一ID值就方便多了。在构造方法里，调用UUID.randomUUID()产生一个随机唯一ID值。

Android Studio可能会找到两个同名Date类。使用Option+Return（或Alt+Enter）快捷键手动导入类。在确认应导入哪个版本的Date类时，选择java.util.Date类。

使用默认的Date构造方法初始化Data变量。作为crime的默认发生时间，设置mDate变量值为当前日期。

接下来，为只读成员变量mId生成一个getter方法，为成员变量mTitle、mDate和mSolved生成getter方法和setter方法。右键单击构造方法下面的空白处，选择Generate... → Getter菜单项，然后选择mId变量。再选择Generate... → Getter and Setter为变量mTitle、mDate和mSolved生成getter方法和setter方法，如代码清单7-4所示。

代码清单7-4 已生成的getter方法与setter方法（Crime.java）

```
public class Crime {

    private UUID mId;
    private String mTitle;
    private Date mDate;
    private boolean mSolved;

    public Crime() {
        mId = UUID.randomUUID();
        mDate = new Date();
    }

    public UUID getId() {
        return mId;
    }
```

```
    public String getTitle() {
        return mTitle;
    }

    public void setTitle(String title) {
        mTitle = title;
    }

    public Date getDate() {
        return mDate;
    }

    public void setDate (Date date) {
        mDate = date;
    }

    public boolean isSolved() {
        return mSolved;
    }

    public void setSolved (boolean solved) {
        mSolved = solved;
    }
}
```

以上是本章CriminalIntent模型层及`Crime`类所需的全部代码实现工作。

至此，除了模型层，我们还创建了能够托管支持库版fragment的activity。接下来，继续学习activity托管fragment的具体实现部分。

7.4　托管 UI fragment

为托管UI fragment，activity必须：

- 在其布局中为fragment的视图安排位置；
- 管理fragment实例的生命周期。

7.4.1　fragment 的生命周期

图7-11展示了fragment的生命周期。类似于activity的生命周期，它具有停止、暂停以及运行状态，也拥有可以覆盖的方法，用来在关键节点完成一些任务。可以看到，许多方法对应着activity的生命周期方法。

这种对应非常重要。因为fragment代表activity工作，所以它的状态应该反映activity的状态。显然，fragment需要相对应的生命周期方法来处理activity的工作。

fragment生命周期与activity生命周期的一个关键区别就在于，fragment的生命周期方法由托管activity而不是操作系统调用。操作系统不关心activity用来管理视图的fragment。fragment的使用是activity内部的事情。

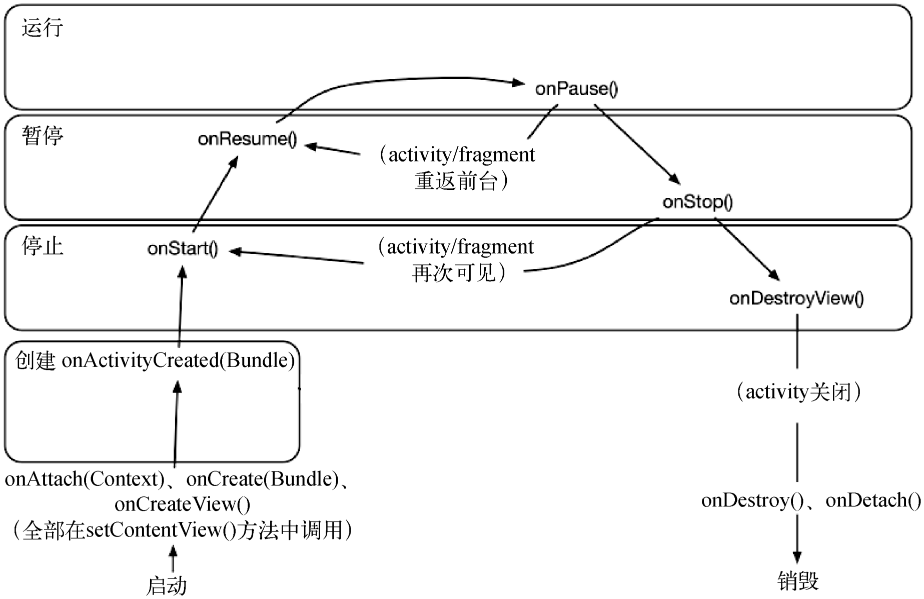


图7-11　fragment的生命周期图解

随着CriminalIntent应用开发的深入，你会看到更多的fragment生命周期方法。

7.4.2　托管的两种方式

activity托管UI fragment有如下两种方式：

- ❑ 在activity布局中添加fragment；
- ❑ 在activity代码中添加fragment。

第一种方式就是使用布局fragment。这种方式简单但不够灵活。在activity布局中添加fragment，就等同于将fragment及其视图与activity的视图绑定在一起，并且在activity的生命周期过程中，无法替换fragment视图。

第二种方式比较复杂，但也是唯一可以动态控制fragment的方式。何时添加fragment以及随后可以完成何种具体任务由你自己定；也可以移除fragment，用其他fragment代替当前fragment，然后重新添加已移除的fragment。

因而，为追求真正灵活的UI设计，就必须通过代码的方式添加fragment。`CrimeActivity`托管`CrimeFragment`就是采用的这种方式。本章稍后会介绍代码的实现细节。现在，先来定义`CrimeActivity`的布局。

7.4.3　定义容器视图

虽然已选择在托管activity代码中添加UI fragment，但还是要在activity视图层级结构中为fragment视图安排位置。在`CrimeActivity`的布局中，该位置就是如图7-12所示的`FrameLayout`。

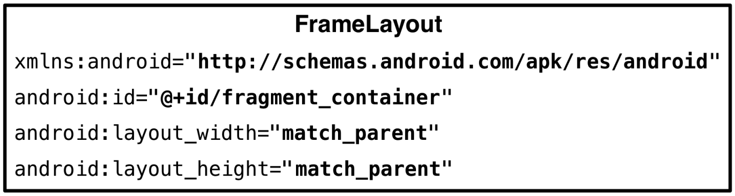


图7-12　CrimeActivity类的fragment托管布局

FrameLayout是服务于CrimeFragment的容器视图。注意该容器视图是个通用性视图，不单单用于CrimeFragment类，你还可以用它托管其他的fragment（后续章节会用到）。

找到CrimeActivity的布局文件res/layout/activity_crime.xml。打开该文件，使用图7-12所示的FrameLayout替换默认布局。完成后的XML文件应如代码清单7-5所示。

代码清单7-5　创建fragment容器布局（activity_crime.xml）

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

注意，当前的activity_crime.xml布局文件仅由一个服务于单个fragment的容器视图组成，但托管activity布局本身也可以非常复杂。除自身组件外，托管activity布局还可定义多个容器视图。

现在预览布局文件，或者运行CriminalIntent应用验证实现代码。不过，CrimeActivity还没有托管任何fragment，因此只能看到一个空的FrameLayout，如图7-13所示。（如果预览窗口无法正确渲染视图，或者有错误发生，请选择Build → Rebuild Project菜单项重建项目。如果仍有问题，可尝试在模拟器或设备上运行应用。预览工具偶尔会有点小问题。）

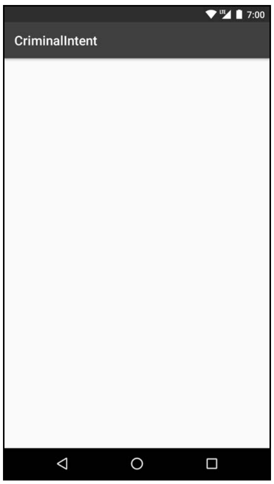


图7-13　一个空的FrameLayout

稍后，我们会编写代码，将fragment的视图放置到FrameLayout中。不过，首先要有一个fragment。（按Android Studio当前对activity的配置，应用顶部的工具栏默认就有。如何定制工具栏，请阅读第13章。）

7.5 创建 UI fragment

创建UI fragment的步骤与创建activity的步骤相同：

- ❑ 定义用户界面布局文件；
- ❑ 创建fragment类并设置其视图为定义的布局；
- ❑ 编写代码以实例化组件。

7.5.1 定义 CrimeFragment 的布局

CrimeFragment视图用来显示包含在Crime类实例中的信息。

首先，打开res/values/strings.xml，添加需要的字符串资源，如代码清单7-6所示。

代码清单7-6 添加字符串资源（res/values/strings.xml）

```
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="crime_title_hint">Enter a title for the crime.</string>
    <string name="crime_title_label">Title</string>
    <string name="crime_details_label ">Details</string>
    <string name="crime_solved_label ">Solved</string>
</resources>
```

然后是定义用户界面。CrimeFragment的视图布局包含一个垂直LinearLayout组件，这个组件又含有5个子组件：两个TextView组件、一个EditText组件、一个Button组件和一个CheckBox组件。

要创建布局文件，在项目工具窗口中，右键单击res/layout文件夹，选择New → Layout resource file菜单项。命名布局文件为fragment_crime.xml。输入LinearLayout作为根元素节点后，单击OK按钮完成创建。

新建文件打开后，查看XML，会发现向导已经添加了LinearLayout。手动添加其余组件，结果如代码清单7-7所示。

代码清单7-7 fragment视图的布局文件（fragment_crime.xml）

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="16dp"
    android:orientation="vertical">

    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="@string/crime_title_label"/>

    <EditText
        android:id="@+id/crime_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/crime_title_hint"/>

    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_details_label"/>

    <Button
        android:id="@+id/crime_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <CheckBox
        android:id="@+id/crime_solved"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_solved_label"/>

</LinearLayout>
```

切换至Design视图，预览已完成的CrimeFragment布局。

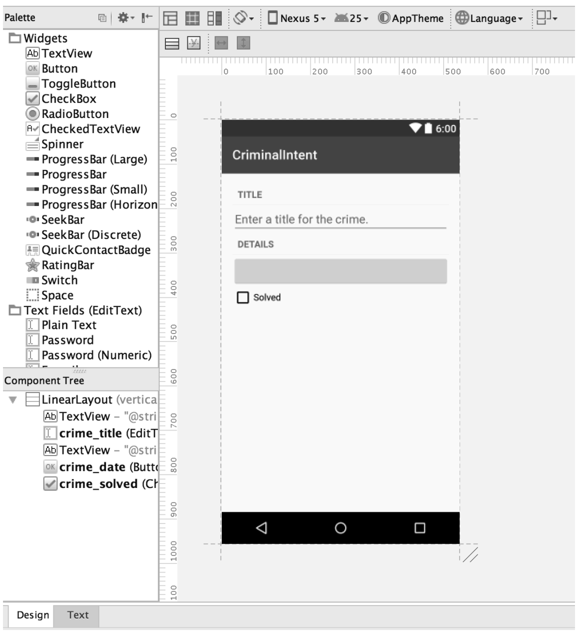


图7-14　预览CrimeFragment布局

（如代码清单7-7所示，fragment_crime.xml布局文件里出现了新语法：`style="?android:listSeparatorTextViewStyle"`。现在不理解没关系，学完9.3.3节，你就会明白了。）

7.5.2　创建 CrimeFragment 类

右键单击com.bignerdranch.android.criminalintent包，选择New → Java Class菜单项。在弹出的新建类对话框中，命名类为`CrimeFragment`，单击OK按钮完成类创建。

修改代码，让`CrimeFragment`类继承`Fragment`类，如代码清单7-8所示。

代码清单7-8　继承Fragment类（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {

}
```

修改代码继承`Fragment`类时，Android Studio会找到两个同名`Fragment`类：`Fragment (android.app)`和`Fragment (android.support.v4.app)`。前者是Android操作系统内置版`Fragment`，后者是支持库版`Fragment`。选择后者，如图7-15所示。

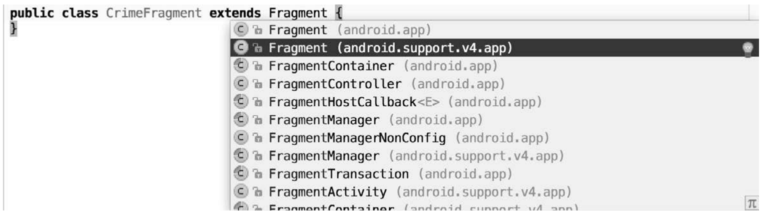


图7-15　选择支持库中的Fragment类

完成后的代码应该和代码清单7-9一样。

代码清单7-9　导入支持库版Fragment（CrimeFragment.java）

```
package com.bignerdranch.android.criminalintent;

import android.support.v4.app.Fragment;

public class CrimeFragment extends Fragment {

}
```

如果看不到Android Studio的提示框，或者误导入了`android.app.Fragment`类，请先删除导入语句，使用Option+Return（或Alt+Enter）快捷键手动重新导入。千万不要搞错，我们需要的是支持库版`Fragment`。

1. 实现fragment生命周期方法

`CrimeFragment`类是与模型及视图对象交互的控制器，用于显示特定crime的明细信息，并在用户修改这些信息后立即进行更新。

在GeoQuiz应用中，activity通过其生命周期方法完成了大部分逻辑控制工作。而在CriminalIntent应用中，这些工作是由fragment的生命周期方法完成的。fragment的许多生命周期方法对应着我们熟知的`Activity`方法，如`onCreate(Bundle)`方法。

在CrimeFragment.java中，新增一个`Crime`实例成员变量，实现`Fragment.onCreate(Bundle)`方法，如代码清单7-10所示。

实现覆盖方法时，Android Studio能够提供便利。在定义`onCreate(Bundle)`方法的过程中，输入方法名的第一个字母时，Android Studio会弹出建议方法清单，如图7-16所示。

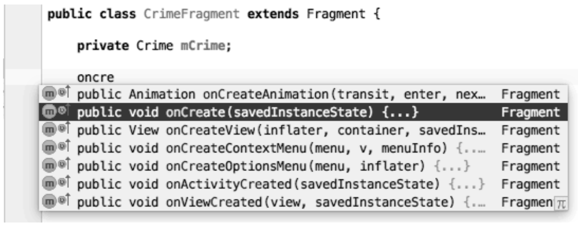


图7-16 覆盖`onCreate(Bundle)`方法

按回车键选择`onCreate(Bundle)`方法，Android Studio会自动创建方法存根。更新代码创建一个新crime，结果如代码清单7-10所示。

代码清单7-10 覆盖`Fragment.onCreate(Bundle)`方法（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }

}
```

以上实现代码中，有以下几点值得一说。

首先，`Fragment.onCreate(Bundle)`是公共方法，而`Activity.onCreate(Bundle)`是受保护方法。`Fragment.onCreate(Bundle)`方法及其他`Fragment`生命周期方法必须是公共方法，因为托管fragment的activity要调用它们。

其次，类似于activity，fragment同样具有保存及获取状态的bundle。如同使用`Activity.onSaveInstanceState(Bundle)`方法那样，你也可以根据需要覆盖`Fragment.onSaveInstanceState(Bundle)`方法。

另外，fragment的视图并没有在`Fragment.onCreate(Bundle)`方法中生成。虽然我们在该方法中配置了fragment实例，但创建和配置fragment视图是另一个`Fragment`生命周期方法完成的：

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState)
```

该方法实例化fragment视图的布局，然后将实例化的View返回给托管activity。LayoutInflater及ViewGroup是实例化布局的必要参数。Bundle用来存储恢复数据，可供该方法从保存状态下重建视图。

在CrimeFragment.java中，添加onCreateView(...)方法的实现代码，从fragment_crime.xml布局中实例化并返回视图，如代码清单7-11所示。

代码清单7-11　覆盖onCreateView(...)方法（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        return v;
    }
}
```

在onCreateView(...)方法中，fragment的视图是直接通过调用LayoutInflater.inflate(...)方法并传入布局的资源ID生成的。第二个参数是视图的父视图，我们通常需要父视图来正确配置组件。第三个参数告诉布局生成器是否将生成的视图添加给父视图。这里，传入了false参数，因为我们将以代码的方式添加视图。

2. 在fragment中实例化组件

现在来生成fragment中的EditText、CheckBox和Button组件。它们也是在onCreateView(...)方法里实例化。

首先处理EditText组件。视图生成后，引用并为它添加对应的监听器方法，如代码清单7-12所示。

代码清单7-12　生成并使用EditText组件（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);

        mTitleField = (EditText)v.findViewById(R.id.crime_title);
        mTitleField.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(
```

```
                    CharSequence s, int start, int count, int after) {
                    // This space intentionally left blank
                }

                @Override
                public void onTextChanged(
                    CharSequence s, int start, int before, int count) {
                    mCrime.setTitle(s.toString());
                }

                @Override
                public void afterTextChanged(Editable s) {
                    // This one too
                }
            });

            return v;
        }
    }
```

Fragment.onCreateView(...)方法中的组件引用几乎等同于Activity.onCreate(Bundle)方法的处理。唯一的区别是，你调用了fragment视图的View.findViewById(int)方法。以前使用的Activity.findViewById(int)方法十分便利，能够在后台自动调用View.findViewById(int)方法，而Fragment类没有对应的便利方法，因此必须手动调用。

fragment中监听器方法的设置和activity中完全一样。创建实现TextWatcher监听器接口的匿名内部类，如代码清单7-12所示。TextWatcher有三个方法，不过现在只需关注其中的onTextChanged(...)方法。

在onTextChanged(...)方法中，调用CharSequence（代表用户输入）的toString()方法。该方法最后返回用来设置Crime标题的字符串。

接下来处理Button组件，让它显示crime的发生日期，如代码清单7-13所示。

代码清单7-13　设置Button文字（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        ...
        mDateButton = (Button) v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);

        return v;
    }
}
```

禁用Button按钮，确保它不会响应用户的点击。按钮应处于灰色状态，这样用户一看就知道，按钮是不可以按的。第12章中，Button按钮会重新启用，并允许用户随意选择crime日期。

最后处理CheckBox组件。引用它并设置监听器，根据用户操作，更新mSolved状态，如代码清单7-14所示。

代码清单7-14 监听CheckBox的变化（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckBox;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        ...
        mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
        mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
                    boolean isChecked) {
                mCrime.setSolved(isChecked);
            }
        });

        return v;
    }
}
```


敲完以上代码，点击OnCheckedChangeListener：

```
mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener()
```

然后，使用Option+Return或Alt+Enter组合键添加包导入语句。Android Studio会提供两个选择，选android.widget.CompoundButton。

取决于你的Android Studio版本，代码自动补全功能可能会插入CompoundButton.OnChecked-ChangeListener,也可能还是原来的OnCheckedChangeListener。两种形式都没问题。但为了代码统一，建议保持OnCheckedChangeListener这种形式。这时，可以按Option+Return或Alt+Enter组合键，然后选择Add on demand static import for 'android.widget.CompoundButton'，如图7-17所示。这样，代码就更新得和代码清单7-14一样了。

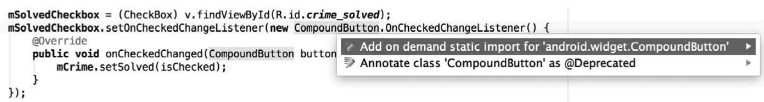


图7-17 添加静态引入

CrimeFragment类的代码实现部分完成了，但现在还不能运行应用查看用户界面和检验代码。

这是因为fragment自己无法在屏幕上显示视图，怎么办？把CrimeFragment添加给CrimeActivity。

7.6　向 FragmentManager 添加 UI fragment

在 Honeycomb 引入 Fragment 类的时候，为协同工作，Activity 类中相应添加了 FragmentManager类。FragmentManager类负责管理fragment并将它们的视图添加到activity的视图层级结构中，如图7-18所示。

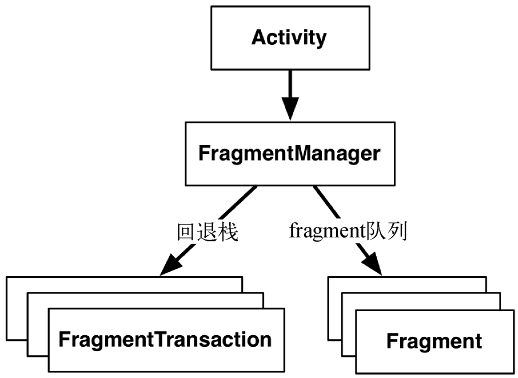


图7-18　FragmentManager图解

FragmentManager类具体管理：

- fragment队列；
- fragment事务回退栈（稍后会学习）。

在CriminalIntent应用中，你只需要关心FragmentManager管理的fragment队列。

要以代码的方式将fragment添加给activity，需要直接调用activity的FragmentManager。首先是获取FragmentManager本身。在CrimeActivity.java中，在onCreate(Bundle)方法中添加代码取得FragmentManager，如代码清单7-15所示。

代码清单7-15　获取FragmentManager（CrimeActivity.java）

```
public class CrimeActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
    }
}
```

如果添完代码遇到错误，记得检查导入语句，看是否已导入支持库版本的FragmentManager类。

因为使用了支持库及AppCompatActivity类，所以这里调用了getSupportFragmentManager()方法。如果不考虑旧版本的兼容性问题，可直接继承Activity类并调用getFragmentManager()方法。

7.6.1 fragment 事务

获取FragmentManager之后，再获取一个fragment交给它管理，如代码清单7-16所示。（现在只需对照添加，稍后会逐行解读代码。）

代码清单7-16 添加一个CrimeFragment（CrimeActivity.java）

```
public class CrimeActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

以上代码中，获取fragment不难理解。add(...)方法及其相关代码才是重点。这段代码创建并提交了一个fragment事务：

```
if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragment_container, fragment)
        .commit();
```

fragment事务被用来添加、移除、附加、分离或替换fragment队列中的fragment。这是使用fragment动态组装和重新组装用户界面的关键。FragmentManager管理着fragment事务回退栈。

FragmentManager.beginTransaction()方法创建并返回FragmentTransaction实例。FragmentTransaction类支持流接口（fluent interface）的链式方法调用，以此配置FragmentTransaction再返回它。因此，以上灰底代码可解读为："创建一个新的fragment事务，执行一个fragment添加操作，然后提交该事务。"

add(...)方法是整个事务的核心，它有两个参数：容器视图资源ID和新创建的CrimeFragment。容器视图资源ID你应该很熟悉了，它是定义在activity_crime.xml中的FrameLayout组件的资源ID。容器视图资源ID的作用有：

- 告诉FragmentManager，fragment视图应该出现在activity视图的什么位置；
- 唯一标识FragmentManager队列中的fragment。

如需从FragmentManager中获取CrimeFragment，使用容器视图资源ID就行了：

```
FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragment_container);

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragment_container, fragment)
        .commit();
}
```

FragmentManager使用FrameLayout组件的资源ID识别CrimeFragment，这看上去可能有点怪。但实际上，使用容器视图资源ID识别UI fragment就是FragmentManager的一种内部实现机制。如果要向activity添加多个fragment，通常就需要分别为每个fragment创建具有不同ID的不同容器。

现在从头至尾对代码清单7-16中的新增代码作一个总结。

首先，使用R.id.fragment_container的容器视图资源ID，向FragmentManager请求并获取fragment。如果要获取的fragment在队列中，FragmentManager就直接返回它。

为什么要获取的fragment可能有了呢？前面说过，设备旋转或回收内存时，Android系统会销毁CrimeActivity，而后重建时，会调用CrimeActivity.onCreate(Bundle)方法。activity被销毁时，它的FragmentManager会将fragment队列保存下来。这样，activity重建时，新的FragmentManager会首先获取保存的队列，然后重建fragment队列，从而恢复到原来的状态。

当然，如果指定容器视图资源ID的fragment不存在，则fragment变量为空值。这时应该新建CrimeFragment，并启动一个新的fragment事务，将新建fragment添加到队列中。

CrimeActivity目前托管着CrimeFragment。运行CriminalIntent应用验证这一点，应该可以看到定义在fragment_crime.xml中的视图，如图7-19所示。



图7-19　CrimeActivity托管的CrimeFragment视图

7.6.2　FragmentManager 与 fragment 生命周期

掌握了FragmentManager的基本使用后，有必要重新审视fragment的生命周期，如图7-20所示。

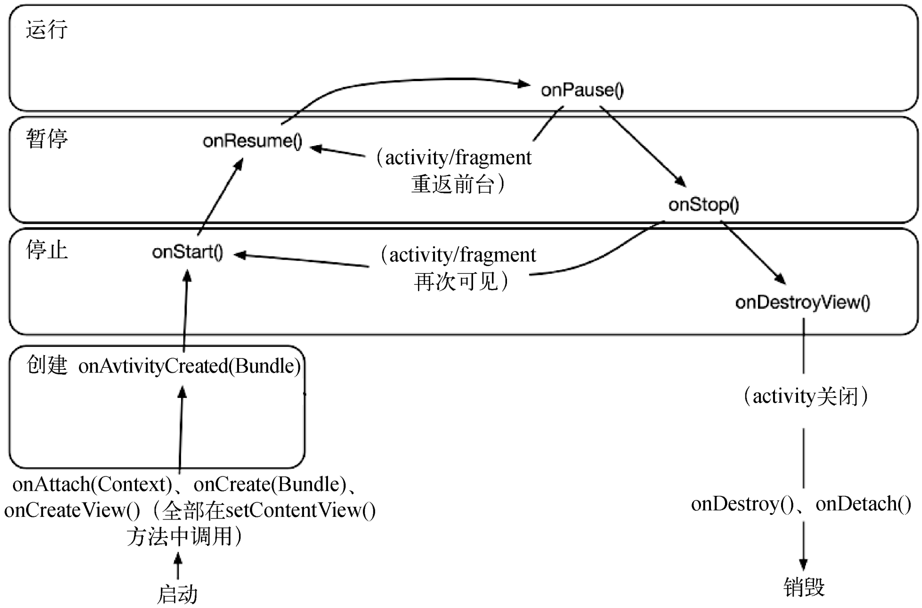


图7-20　再探fragment生命周期

activity的FragmentManager负责调用队列中fragment的生命周期方法。添加fragment供FragmentManager管理时，onAttach(Context)、onCreate(Bundle)和onCreateView(...)方法会被调用。

托管activity的onCreate(Bundle)方法执行后，onActivityCreated(Bundle)方法也会被调用。因为是在CrimeActivity.onCreate(Bundle)方法中添加CrimeFragment，所以fragment被添加后，该方法会被调用。

在activity处于运行状态时，添加fragment会发生什么呢？这种情况下，FragmentManager立即驱赶fragment，调用一系列必要的生命周期方法，快速跟上activity的步伐（与activity的最新状态保持同步）。例如，向处于运行状态的activity中添加fragment时，以下fragment生命周期方法会被依次调用：onAttach(Context)、onCreate(Bundle)、onCreateView(...)、onActivityCreated(Bundle)、onStart()以及onResume()。

一旦追上，托管activity的FragmentManager就会边接收操作系统的调用指令，边调用其他生命周期方法，让 fragment与activity的状态取得一致。

7

7.7　采用 fragment 的应用架构

设计应用时，正确使用fragment非常重要。然而，许多开发者学习了fragment之后，为了复用组件，只要可能，就直接使用 fragment。这实际是在滥用fragment。

fragment是用来封装关键组件以方便复用。这里所说的关键组件，是针对应用的整个屏幕来讲的。如果单屏就使用大量fragment，不仅应用代码充斥着fragment事务处理，模块的职责分工也会不清晰。如果有很多零碎小组件要复用，比较好的架构设计是使用定制视图(使用`View`子类)。

总之，一定要合理使用fragment。实践证明，应用单屏最多使用2 ~ 3个fragment，如图7-21所示。



图7-21　少就是多的哲学

使用 fragment 的理由

自本章开始，无论应用多么简单，你都会看到 fragment的身影。这貌似有点激进，要知道，后续章节的很多应用不用fragment也行。用户界面可以只使用activity来创建和管理，而且，这样做的话，代码量会更少。

然而，我们认为，这是实际开发中比较实用的模式，建议尽早适应。

有人可能觉得，在应用开发初期暂不使用fragment，等到需要时再添加它会好一些。极限编程方法论中有个YAGNI原则。YAGNI (You Aren't Gonna Need It) 的意思是“你不会需要它”，该原则鼓励大家不要去实现那些有可能需要的东西。为什么呢？因为你不会需要它。因此，YAGNI原则等于在说，可用可不用，那就不要用fragment了。

不幸的是，经验表明，后期添加fragment就如同掉泥坑。从activity管理用户界面调整到由activity托管UI fragment虽然不难，但会有一大堆恼人的问题等着你。你也可能会想让部分用户界面仍由activity管理，部分用户界面改用fragment管理，这只会让事情更糟。哪些不改，哪些要改，光理清这些就够你头痛的了。显然，从一开始就使用fragment更容易，既不用返工，也不会出现理不清哪个部分使用了哪种视图控制风格这种事了。

因而，对于fragment，我们坚持AUF（Always Use Fragments）原则，即“总是使用fragment”。不值得为使用fragment还是activity伤脑筋，相信我们，总是使用fragment！

7.8　深入学习：fragment 与支持库

在本章中，为使用支持库版fragment，CriminalIntent项目引入了AppCompat库。然而，这个库自身并没有实现fragment功能，它依赖support-v4库。所以，真正实现fragment功能的是support-v4库。

Google为开发者提供的支持库有很多，如support-v4、appcompat-v7和recyclerview-v7等。早期，支持库只有一个，说支持库就是指support-v4库。后来，各种功能越加越多，这个库不断膨胀，最终不可避免地成了一个大杂烩。于是，Google决定另起炉灶，然后就有了上面提到的各种支持库。

前面说过，support-v4库支持实现fragment功能。你可以在该库中找到`android.support.v4.app.Fragment`的源码。support-v4库里也有一个`Activity`子类：`FragmentActivity`。要使用支持库版fragment，应用的activity必须继承`FragmentActivity`。

如图7-22所示，`AppCompatActivity`是`FragmentActivity`的子类。CriminalIntent项目引入了AppCompat库，所以应用能使用支持库版fragment。如果只想用support-v4库也可以，只要在项目里引入它，然后把各个activity的父类从`AppCompatActivity`改为`FragmentActivity`就行了。改来改去好麻烦。没关系，和本书一样，就用AppCompat支持库吧。如今，大多数开发人员都在用它。第13章还会介绍AppCompat支持库的其他功能，敬请期待。

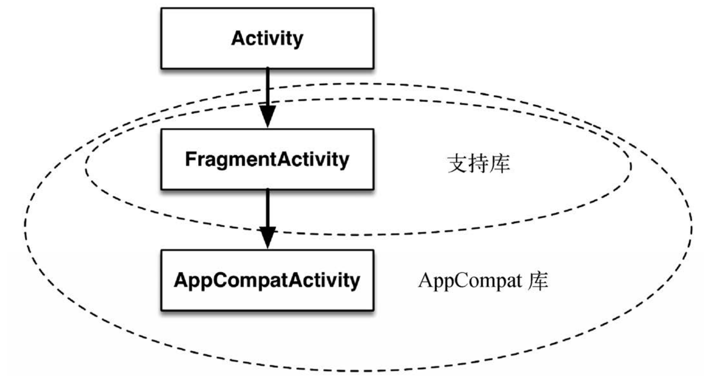


图7-22　`AppCompatActivity`继承树

7.9　深入学习：为什么优先使用支持库版 fragment

本书坚持使用支持库版fragment。这似乎不是一条寻常路。毕竟，Google提供支持库版fragment的本意是方便在不支持该API的旧版本上使用fragment。要知道，大多数开发人员如今都只用操作系统内置版fragment。

坚持的理由是什么？我们认为，支持库版fragment使用最方便。任何时候，只要想升级支持库版fragment的话，只需要下载升级包，重新编译发布一个新版本应用就行了。Google每年会多次更新支持库，并借此引入新特性、修复bug。要享受这些好处，升级应用的支持库版本就行了。

举个例子，Google自Android 4.2开始支持fragment嵌套使用（在fragment中托管fragment）。如果基于操作系统内置版fragment开发，并且面向Android 4.0及以上版本的设备，那么应用就无法使用这个新特性了。假如用了支持库版fragment，就能轻松升级应用的支持库版本，享用fragment嵌套新特性（设备内存要足够大哦）。

此外，使用支持库版fragment没有显著的缺点。就功能实现来讲，它和操作系统内置版本没什么不同。非要挑毛病的话，那就是导入支持库包会占用额外空间。不过考虑到上述诸多优点，牺牲不到1M的空间并不算什么。

本书强调实用，多年的开发实践表明，Android支持库就是无冕之王。

当然，如果你有自己的想法，非要用操作系统内置版fragment，也可以。

要使用标准库里的fragment，需对项目做以下改动。

- 弃用`FragmentActivity`类，改用标准库中的`Activity`类（`android.app.Activity`）。它默认支持在API 11级或更高版本系统中使用fragment。
- 弃用`android.support.v4.app.Fragment`类，改用`android.app.Fragment`类。
- 弃用`getSupportFragmentManager()`方法，改用`getFragmentManager()`方法获取`FragmentManager`。

第 8 章 使用RecyclerView显示列表

当前，CriminalIntent应用的模型层仅包含一个Crime实例。本章，我们将更新CriminalIntent应用以支持显示crime列表。列表会显示每个Crime实例的标题及其发生日期，如图8-1所示。



图8-1　crime列表

图8-2是CriminalIntent应用在本章的整体规划图。

应用模型层将新增一个CrimeLab对象，该对象是一个数据集中存储池，用来存储Crime对象。

显示crime列表需在应用控制器层新增一个activity和一个fragment：CrimeListActivity和CrimeListFragment。

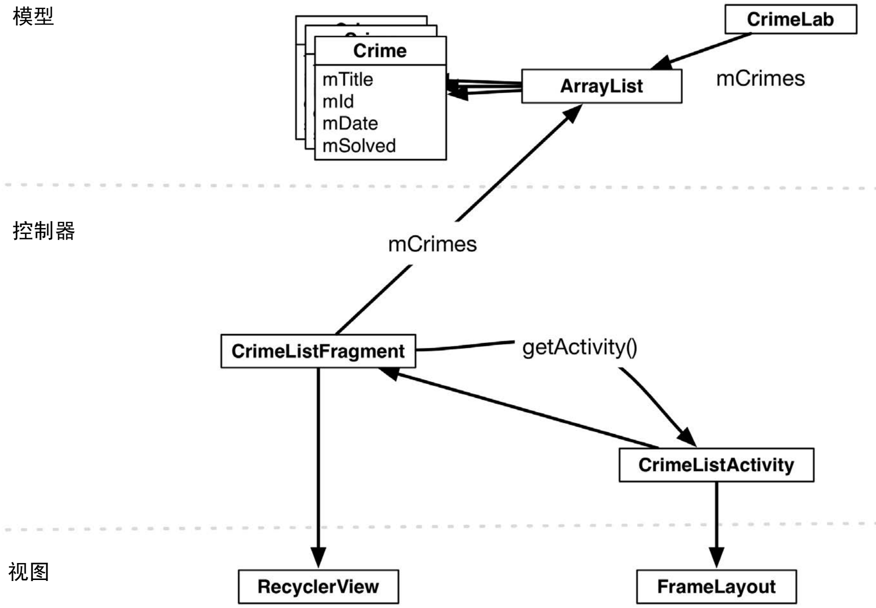


图8-2　CriminalIntent应用对象图

（图8-2中怎么没有CrimeActivity和CrimeFragment呢？它们是与明细视图相关的类，所以这里没有显示。在第10章中，我们将学习如何关联CriminalIntent应用的列表视图和明细视图。）

在图8-2中，也可以看到与CrimeListActivity和CrimeListFragment关联的视图对象。activity视图由包含fragment的FrameLayout组成。fragment视图由一个RecyclerView组成。稍后会介绍RecyclerView类。

8.1　升级 CriminalIntent 应用的模型层

首先，我们来升级应用的模型层，从容纳单个Crime对象变为可容纳一组Crime对象。

单例与数据集中存储

crime数组对象将存储在一个单例里。单例是特殊的Java类，在创建实例时，一个单例类仅允许创建一个实例。

应用能在内存里活多久，单例就能活多久。因此将对象列表保存在单例里的话，就能随时获取crime数据，不管activity和fragment的生命周期怎么变化。使用单例还要注意一点：Android从内存里清除应用时，单例对象也会随之消失。虽然CrimeLab单例不是数据持久保存的好方案，但它确实能保证仅拥有一份crime数据，并且能让控制器层类间的数据传递更容易。（第14章会介绍如何持久化保存数据。）

（要进一步了解单例类，请参见8.7节。）

要创建单例，需创建一个带有私有构造方法及get()方法的类。如果实例已存在，get()方法就直接返回它；如果实例还不存在，get()方法就会调用构造方法创建它。

右键单击com.bignerdranch.android.criminalintent类包，选择New → Java Class菜单项。在随后出现的对话框中，命名类为CrimeLab，然后单击OK按钮。

在打开的CrimeLab.java文件中，编码实现CrimeLab类为带有私有构造方法和get()方法的单例，如代码清单8-1所示。

代码清单8-1　创建单例（CrimeLab.java）

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    public static CrimeLab get(Context context) {
        if (sCrimeLab == null) {
            sCrimeLab = new CrimeLab(context);
        }
        return sCrimeLab;
    }

    private CrimeLab(Context context) {

    }
}
```

以上代码有几点值得一说。

首先，注意sCrimeLab变量的s前缀。这是Android开发的命名约定，一看到此前缀，我们就知道sCrimeLab是一个静态变量。

其次，再来看CrimeLab的私有构造方法。显然，其他类无法创建CrimeLab对象，除非调用get()方法。

最后，在get()方法里，我们传入的是Context对象（第14章会用到）。

下面，我们往CrimeLab中存储Crime对象。在CrimeLab的构造方法里，创建一个空List用来保存Crime对象。此外，再添加两个方法：getCrimes()和getCrime(UUID)。前者返回数组列表，后者返回带指定ID的Crime对象，如代码清单8-2所示。

代码清单8-2　创建可容纳Crime对象的List（CrimeLab.java）

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;

    public static CrimeLab get(Context context) {
        ...
    }

    private CrimeLab(Context context) {
        mCrimes = new ArrayList<>();
```

8

```
    }

    public List<Crime> getCrimes() {
        return mCrimes;
    }

    public Crime getCrime(UUID id) {
        for (Crime crime : mCrimes) {
            if (crime.getId().equals(id)) {
                return crime;
            }
        }

        return null;
    }
}
```

List<E>是一个泛型类，支持存放特定数据类型的有序列表对象，拥有获取、新增和删除列表元素的方法。常见的List实现有ArrayList（使用常规Java数组存储列表元素）。

既然mCrimes含有ArrayList，而ArrayList也是一个List，那么对于mCrimes来说，ArrayList和List都是有效的类型。有鉴于此，推荐在声明变量的时候使用List接口类型。这样，若有需要，还可以方便地使用其他List实现，如LinkedList。

mCrimes实例化语句使用了Java 7引入的<>符号。该符号告诉编译器，List中的元素类型可以基于变量声明传入的抽象参数来确定。这里，因为变量声明语句private List<Crime> mCrimes;中指定了Crime参数，所以编译器可据此推测出ArrayList里可放入Crime对象。（Java 7 之前，必须这么写：mCrimes = new ArrayList<Crime>();。）

最后，新建List将包含用户自建的Crime，用户可自由存取它们。现在，先批量存入100个毫无个性的Crime对象，如代码清单8-3所示。

代码清单8-3　生成100个crime（CrimeLab.java）

```
private CrimeLab(Context context) {
    mCrimes = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Crime crime = new Crime();
        crime.setTitle("Crime #" + i);
        crime.setSolved(i % 2 == 0); // Every other one
        mCrimes.add(crime);
    }
}
```

这样，一个满载100个crime数据的模型层诞生了。

8.2　使用抽象 activity 托管 fragment

创建托管CrimeListFragment的CrimeListActivity类之前，首先为CrimeListActivity创建视图。

8.2.1 通用型 fragment 托管布局

对于CrimeListActivity，我们仍可以使用定义在activity_crime.xml文件中的布局（代码清单8-4）。该布局提供了一个放置fragment的FrameLayout容器视图，其中的fragment可在activity中使用代码获取。

代码清单8-4 通用的布局定义文件activity_crime.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

activity_crime.xml布局文件并没有特别指定fragment，任何activity托管fragment的场景，都可以使用它。下面，为了让该布局更加通用，重命名它为activity_fragment.xml。

在项目工具窗口中，右键单击res/layout/activity_crime.xml文件。（注意是单击activity_crime.xml文件，而不是fragment_crime.xml。）

在弹出菜单里，选择Refactor → Rename...菜单项，将activity_crime.xml改名为activity_fragment.xml。

Android Studio应该自动更新引用。如果看到CrimeActivity.java代码有错，则需要在CrimeActivity文件中手动更新引用代码，如代码清单8-5所示。

代码清单8-5 为CrimeActivity更新布局文件引用（CrimeActivity.java）

```
public class CrimeActivity extends AppCompatActivity {
    /** Called when the activity is first created. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

8.2.2 抽象 activity 类

可以复用CrimeActivity的代码来创建CrimeListActivity类。回顾一下前面写的Crime-

Activity类，它的实现代码简单且近乎通用（已复制为代码清单8-6）。事实上，CrimeActivity类的代码唯一不通用的地方是CrimeFragment类在添加到FragmentManager之前的实例化部分。

代码清单8-6　近乎通用的CrimeActivity类（CrimeActivity.java）

```
public class CrimeActivity extends AppCompatActivity {
    /** Called when the activity is first created. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

本书中，几乎每次新建activity都需要这样一段代码。为避免重复，我们将这些重复代码封装为抽象类。

在CriminalIntent类包里创建一个名为SingleFragmentActivity的抽象类。设置超类为AppCompatActivity类，如代码清单8-7所示。

代码清单8-7　创建一个Activity抽象类（SingleFragmentActivity.java）

```
public abstract class SingleFragmentActivity extends AppCompatActivity {

}
```

然后，按照代码清单8-8添加相关代码。可以看到，除了灰底部分，其余代码和原来的CrimeActivity代码完全一样。

代码清单8-8　添加一个通用超类（SingleFragmentActivity.java）

```
public abstract class SingleFragmentActivity extends AppCompatActivity {

    protected abstract Fragment createFragment();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
```

```
                fragment = createFragment();
                fm.beginTransaction()
                    .add(R.id.fragment_container, fragment)
                    .commit();
            }
        }
    }
```

在以上代码里，我们设置从activity_fragment.xml布局里实例化activity视图。然后在容器中查找`FragmentManager`里的fragment。如果找不到，就新建fragment并将其添加到容器中。

代码清单8-8与`CrimeActivity`代码唯一的区别就是，为了实例化新的fragment，我们新增了名为`createFragment()`的抽象方法。`SingleFragmentActivity`的子类会实现该方法，来返回由activity托管的fragment实例。

1. 使用抽象类

下面来试试使用`CrimeActivity`抽象类。首先将它的超类改为`SingleFragmentActivity`。然后，删除`onCreate(Bundle)`方法，再添加代码清单8-9所示的`createFragment()`方法。

代码清单8-9　清理`CrimeActivity`类（CrimeActivity.java）

```
public class CrimeActivity extends AppCompatActivity SingleFragmentActivity {
    /** Called when the activity is first created. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }

    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
    }
}
```

2. 新建控制类

现在，新建两个控制类：`CrimeListActivity`和`CrimeListFragment`。

右键单击com.bignerdranch.android.criminalintent包，选择New→Java Class菜单项，在弹出对话框中，命名类为`CrimeListActivity`。

修改`CrimeListActivity`类的超类为`SingleFragmentActivity`类，并实现`createFragment()`

方法，如代码清单8-10所示。

代码清单8-10　实现CrimeListActivity（CrimeListActivity.java）

```
public class CrimeListActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }
}
```

如果新建类时工具默认加入了其他方法，如onCreate，请手动删除。让SingleFragment-Activity做它自己的工作，保持CrimeListActivity类尽量简单。

接下来是创建CrimeListFragment类。

再次右键单击com.bignerdranch.android.criminalintent包，选择New → Java Class菜单项，在弹出对话框中，命名类为CrimeListFragment，如代码清单8-11所示。

代码清单8-11　实现CrimeListFragment（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {

      // Nothing yet

}
```

如代码清单8-11所示，CrimeListFragment类现在是个空结构。稍后再来处理它。

随着后续章节的深入学习，相信你会发现，使用SingleFragmentActivity抽象类可大大减少代码输入量，节约开发时间。现在，activity代码看起来很简洁。

3. 在配置文件中声明CrimeListActivity

CrimeListActivity创建完成后，记得在配置文件中声明它。另外，CriminalIntent应用启动后，用户看到的主界面应该是crime列表，因此还要配置CrimeListActivity为launcher activity。

如代码清单8-12所示，在manifest配置文件中，首先声明CrimeListActivity，然后删除CrimeActivity的launcher activity配置，改配CrimeListActivity为launcher activity。

代码清单8-12　声明CrimeListActivity为launcher activity（AndroidManifest.xml）

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity android:name=".CrimeListActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".CrimeActivity">
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
```

现在，CrimeListActivity是launcher activity。运行CriminalIntent应用，会看到CrimeList-Activity的FrameLayout托管了一个无内容的CrimeListFragment，如图8-3所示。

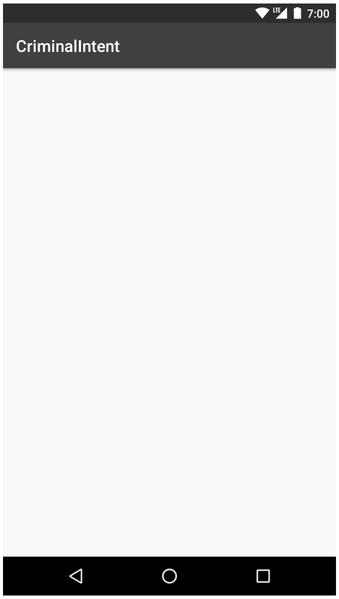


图8-3　没有内容的CrimeListActivity用户界面

8.3　RecyclerView、ViewHolder 和 Adapter

我们需要CrimeListFragment向用户展示crime列表，这就要用到RecyclerView类。

RecyclerView是ViewGroup的子类，每一个列表项都是作为一个View子对象显示的。这些View子对象可简单可复杂，这取决于列表项要显示些什么。

首先来实现简易版的列表项显示，即每个列表项只显示Crime的标题和日期，并且View对象是一个包含两个TextView的LinearLayout，如图8-4所示。

在图8-4中，我们可以看到12行视图View。稍后，升级版CriminalIntent应用能支持滑动屏幕查看所有100个crime项。这是不是意味着要准备100个视图View呢？大声说“不”吧，因为有RecyclerView。

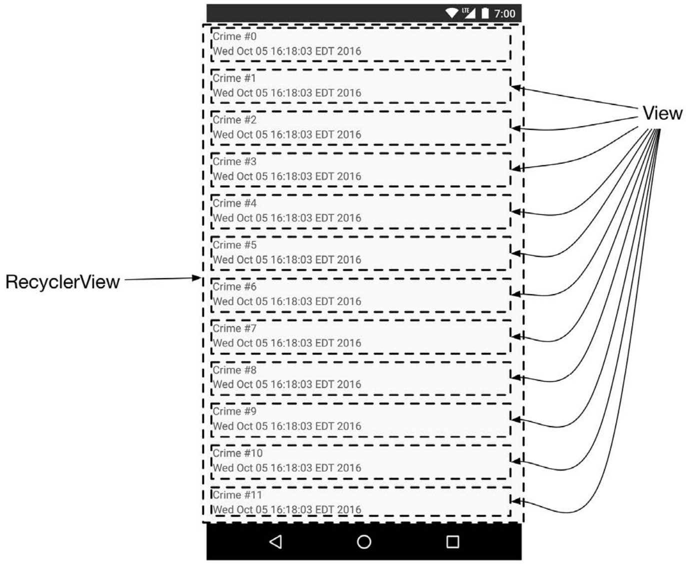


图8-4　带有子View的RecyclerView

一次为所有列表项创建View很容易搞垮应用。可以想象，真实的应用要显示的列表项远不止100个，且要显示更为复杂的内容。另外，在屏幕上显示单个crime的话，单个View也就够了。因此，完全没必要同时准备100个View，按需创建视图对象才是比较合理的解决方案。

RecyclerView就是这么做的。它只创建刚好充满屏幕的12个View，而不是100个。用户滑动屏幕切换视图时，上一个视图会回收利用。顾名思义，RecyclerView所做的就是回收再利用，循环往复。

8.3.1　ViewHolder 和 Adapter

RecyclerView的任务仅限于回收和定位屏幕上的View。列表项View能够显示数据还离不开另外两个类的支持：ViewHolder子类和Adapter子类。ViewHolder要做的事很少，首先介绍它。顾名思义，ViewHolder只做一件事：容纳View视图（如图8-5所示）。

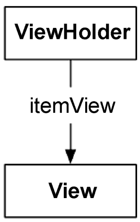


图8-5　没什么地位的ViewHolder

ViewHolder要做的事确实够少了。典型的ViewHolder子类如代码清单8-13所示。

代码清单8-13 典型的ViewHolder子类

```
public class ListRow extends RecyclerView.ViewHolder {
    public ImageView mThumbnail;

    public ListRow(View view) {
        super(view);

        mThumbnail = (ImageView) view.findViewById(R.id.thumbnail);
    }
}
```

你可以创建ListRow来获取自定义的mThumbnail和RecyclerView.ViewHolder超类传入的itemView，如代码清单8-14所示。ViewHolder为itemView而生：它引用着传给super(view)的整个View视图。

代码清单8-14 ViewHolder的使用示例

```
ListRow row = new ListRow(inflater.inflate(R.layout.list_row, parent, false));
View view = row.itemView;
ImageView thumbnailView = row.mThumbnail;
```

RecyclerView自身不会创建视图，它创建的是ViewHolder，而ViewHolder引用着itemView，如图8-6所示。

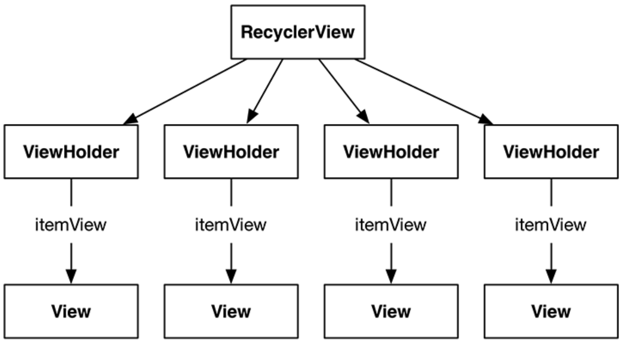


图8-6 ViewHolder配合RecyclerView使用

如果处理的是简单视图，ViewHolder的工作也会相对简单。如果是复杂视图，ViewHolder就得处理不同部分的itemView，以简单高效地展示Crime项。稍后，自定义复杂视图时，你就能一窥究竟。

Adapter

图8-6做了简化，实际上隐藏了一些信息。RecyclerView自己不创建ViewHolder。这个任务实际是由Adapter来完成的。Adapter是一个控制器对象，从模型层获取数据，然后提供给RecyclerView显示，是沟通的桥梁。

Adapter负责：

- 创建必要的ViewHolder；
- 绑定ViewHolder至模型层数据。

要创建Adapter，首先要定义RecyclerView.Adapter子类。然后由它封装从CrimeLab获取的crime。

RecyclerView需要显示视图对象时，就会去找它的Adapter。图8-7展示了RecyclerView可能发起的会话。

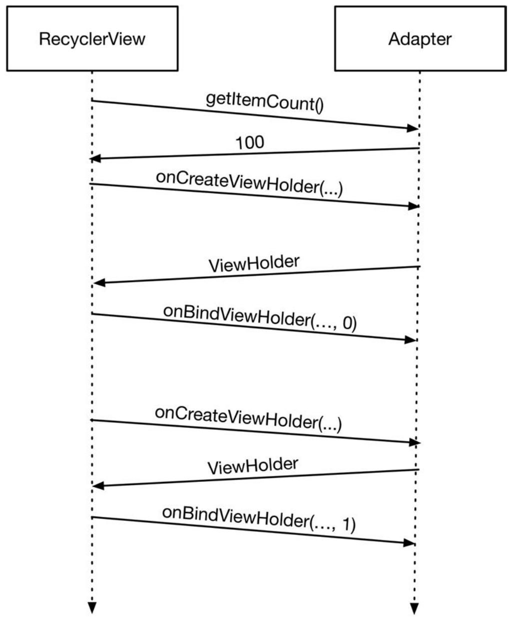


图8-7　生动有趣的RecyclerView-Adapter会话

首先，调用Adapter的getItemCount()方法，RecyclerView询问数组列表中包含多少个对象。

接着，RecyclerView调用Adapter的onCreateViewHolder(ViewGroup, int)方法创建ViewHolder及其要显示的视图。

最后，RecyclerView会传入ViewHolder及其位置，调用onBindViewHolder(ViewHolder, int)方法。Adapter会找到目标位置的数据并将其绑定到ViewHolder的视图上。所谓绑定，就是使用模型数据填充视图。

整个过程执行完毕，RecyclerView就能在屏幕上显示crime列表项了。需要注意的是，相对于onBindViewHolder(ViewHolder, int)方法，onCreateViewHolder(ViewGroup, int)方法的调用并不频繁。一旦有了够用的ViewHolder，RecyclerView就会停止调用onCreateViewHolder(...)方法。随后，它会回收利用旧的ViewHolder以节约时间和内存。

8.3.2 使用 RecyclerView

原理说了这么多，可以动手使用RecyclerView类了。RecyclerView类来自于Google支持库。要使用它，首先要添加RecyclerView依赖库。

单击File → Project Structure....菜单项切换至项目结构窗口，选择左边的app模块，然后单击Dependencies选项页。单击+按钮弹出依赖库添加窗口。

找到并选择recyclerview-v7支持库，单击OK按钮完成依赖库添加，如图8-8所示。

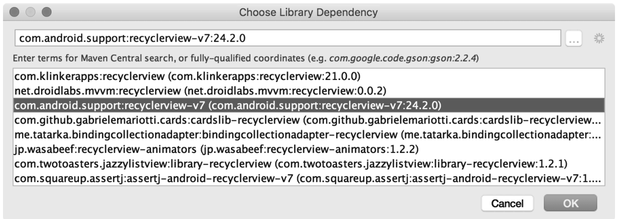


图8-8 添加RecyclerView依赖库

RecyclerView视图需在CrimeListFragment的布局文件中定义。现在创建这个布局文件。右键单击res/layout目录，选择New → Layout resource file菜单项。命名布局文件为fragment_crime_list后单击OK按钮完成。

打开新建的fragment_crime_list.xml布局文件，修改根视图为RecyclerView，并为其配置ID属性，如代码清单8-15所示。

代码清单8-15 在布局文件中添加RecyclerView视图（fragment_crime_list.xml）

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">

</LinearLayout>
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/crime_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

配置完CrimeListFragment的视图，接下来的任务就是视图和fragment的关联。修改CrimeListFragment类文件，使用布局并找到布局中的RecyclerView视图，如代码清单8-16所示。

代码清单8-16 为CrimeListFragment配置视图（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {

    // Nothing yet
    private RecyclerView mCrimeRecyclerView;

    @Override
```

```
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_crime_list, container, false);

        mCrimeRecyclerView = (RecyclerView) view
                .findViewById(R.id.crime_recycler_view);
        mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        return view;
    }

}
```

注意，没有LayoutManager的支持，不仅RecyclerView无法工作，还会导致应用崩溃。所以，RecyclerView视图创建完成后，就立即转交给了LayoutManager对象。

RecyclerView类不会亲自摆放屏幕上的列表项。实际上，摆放的任务被委托给了LayoutManager。除了在屏幕上摆放列表项，LayoutManager还负责定义屏幕滚动行为。因此，没有LayoutManager，RecyclerView也就没法正常工作。

除了一些Android操作系统内置版实现，LayoutManager还有很多第三方库实现版本。我们使用的是LinearLayoutManager类，它支持以竖直列表的形式展示列表项。我们在本书后续章节中还会使用GridLayoutManager类，以网格形式展示列表项。

运行应用，应该还是看不到内容；现在看到的是一个RecyclerView空视图。要显示出crime列表项，还需要完成Adapter和ViewHolder的实现。

8.3.3 列表项视图

如同CrimeFragment的视图，显示在RecyclerView上的列表项有自己的视图层级结构。创建列表项视图布局和创建activity或fragment视图布局没什么不同。在项目工具窗口，右键点击res/layout目录，选择New → Layout resource file菜单项。在弹出的对话框中，命名布局文件为list_item_crime，点击OK按钮完成。

参照图8-9，更新布局文件，添加两个TextView视图组件。

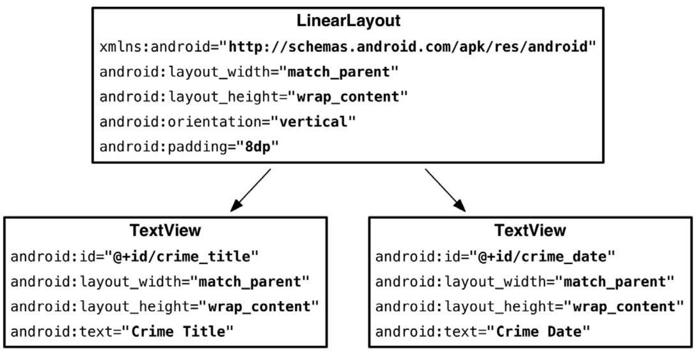


图8-9 更新列表项布局

切换到布局预览模式，应该会看到一个长条形列表项界面。稍后，你就会看到RecyclerView如何使用它。

8.3.4 实现ViewHolder和Adapter

接下来，在CrimeListFragment类中定义ViewHolder内部类，它会实例化并使用list_item_crime布局，如代码清单8-17所示。

代码清单8-17 定义ViewHolder内部类（CrimeListFragment.java）

```
public class CrimelistFragment extends Fragment {
    ...
     private class CrimeHolder extends RecyclerView.ViewHolder {
        public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
            super(inflater.inflate(R.layout.list_item_crime, parent, false));
        }
    }
}
```

在CrimeHolder的构造方法里，我们首先实例化list_item_crime布局，然后传给super(...)方法，也就是ViewHolder的构造方法。基类ViewHolder因而实际引用这个视图。如果你需要，可以在ViewHolder的itemView变量里找到它。

8

定义完ViewHolder，接下来的任务是创建Adapter，如代码清单8-18所示。

代码清单8-18 创建Adapter内部类（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {
    ...
    private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {

        private List<Crime> mCrimes;

        public CrimeAdapter(List<Crime> crimes) {
            mCrimes = crimes;
        }
    }
}
```

（注意，代码清单8-18的代码目前无法编译通过。这个问题稍后会解决。）

需要显示新创建的ViewHolder或让Crime对象和已创建的ViewHolder关联时，RecyclerView会去找Adapter（调用它的方法）。RecyclerView不关心也不了解具体的Crime对象，这是Adapter要做的事。

接下来，在CrimeAdapter中实现三个方法，如代码清单8-19所示。（要自动产生这些覆盖方法存根，可把光标移到extends上，按Option+Return或Alt+Enter组合键，然后选择实现方法，点击OK完成。）

代码清单8-19　武装CrimeAdapter（CrimeListFragment.java）

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
    @Override
    public CrimeHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater layoutInflater = LayoutInflater.from(getActivity());

        return new CrimeHolder(layoutInflater, parent);
    }

    @Override
    public void onBindViewHolder(CrimeHolder holder, int position) {

    }

    @Override
    public int getItemCount() {
        return mCrimes.size();
    }
}
```

RecyclerView需要新的ViewHolder来显示列表项时，会调用onCreateViewHolder方法。在这个方法内部，我们创建一个LayoutInflater，然后用它创建CrimeHolder。

CrimeAdapter必须覆盖onBindViewHolder(...)方法。它现在只是一个空方法，暂时忽略，稍后会完善它。

搞定了Adapter，最后要做的就是将它和RecyclerView关联起来。实现一个设置CrimeList-Fragment用户界面的updateUI方法，该方法创建CrimeAdapter，然后设置给RecyclerView，如代码清单8-20所示。

代码清单8-20　设置Adapter（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_crime_list, container, false);

        mCrimeRecyclerView = (RecyclerView) view
                .findViewById(R.id.crime_recycler_view);
        mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        updateUI();

        return view;
    }
```

```
    private void updateUI() {
        CrimeLab crimeLab = CrimeLab.get(getActivity());
        List<Crime> crimes = crimeLab.getCrimes();

        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    }
    ...
}
```

在稍后的章节中，用户界面的配置会更为复杂，到时会向updateUI()中添加更多内容。

运行CriminalIntent应用，滚动查看RecyclerView视图。应用运行界面如图8-10所示。

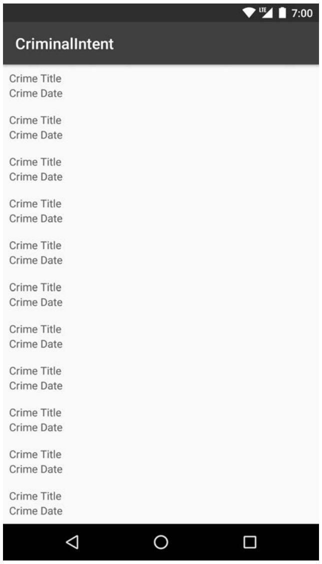


图8-10　多么整齐漂亮的列表项啊

列表项看起来整齐漂亮，但都是一样的面孔啊，RecyclerView先生！左右滑动，上下滚动，都是一个样。

图8-10中有11排数据，也就是说onCreateViewHolder(...)方法被调用了11次。向下滚动，会有更多的CrimeHolder被创建，但达到一定量时，RecyclerView会停止创建新的CrimeHolder。然后它会回收使用那些已滚出屏幕外的CrimeHolder。RecyclerView果然名副其实。

稍后，我们会实现数据绑定，到时，CrimeHolder就能显示真实数据了。

8.4　绑定列表项

所谓绑定，实际就是让Java代码（Crime里的模型数据，或点击监听器）和组件关联起来。目前为止，书中的每一次视图实例化都是绑定。按照过往经验，似乎没必要把绑定任务放入独立

方法。不过，既然CrimeHolder会循环使用，分开处理视图创建和绑定会有好处。

我们把视图绑定工作放入CrimeHolder类里。绑定之前，首先实例化相关组件。由于这是一次性任务，因此直接放在构造方法里处理，如代码清单8-21所示。

代码清单8-21　在构造方法中实例化视图组件（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder {

    private TextView mTitleTextView;
    private TextView mDateTextView;

    public CrimeHolder(layoutinflater inflater, ViewGroup parent) {
        super(inflater.inflate(R.layout.list_item_crime, parent, false));

        mTitleTextView = (TextView) itemView.findViewById(R.id.crime_title);
        mDateTextView = (TextView) itemView.findViewById(R.id.crime_date);
    }
}
```

CrimeHolder还需要一个bind(Crime)方法。每次有新的Crime要在CrimeHolder中显示时，都要调用它一次。现在实现这个方法，如代码清单8-22所示。

代码清单8-22　实现bind(Crime)方法（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder {

    private Crime mCrime;
    ...
    public void bind(Crime crime) {
        mCrime = crime;
        mTitleTextView.setText(mCrime.getTitle());
        mDateTextView.setText(mCrime.getDate().toString());
    }
}
```

现在，只要取到一个Crime，CrimeHolder就会刷新显示TextView标题视图和TextView日期视图。

最后，修改CrimeAdapter类，使用bind(Crime)方法。每次RecyclerView要求CrimeHolder绑定对应的Crime时，都会调用bind(Crime)方法，如代码清单8-23所示。

代码清单8-23　调用bind(Crime)方法（CrimeListFragment.java）

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
    @Override
    public void onBindViewHolder(CrimeHolder holder, int position) {
        Crime crime = mCrimes.get(position);
        holder.bind(crime);
    }
    ...
}
```

再次运行CriminalIntent应用，CrimeHolder能够显示不同的Crime了，如图8-11所示。



图8-11　终于有点样子了

试试看，如果你一通猛滑，列表项滚动得非常流畅。这要归功于onBindViewHolder(...)方法。任何时候，只要有可能，都要确保这个方法轻巧、高效。

8.5 响应点击

为了使RecyclerView锦上添花，列表项应该还能够响应用户的点击。在第10章中，用户点击列表项时，应用支持弹出新界面显示crime明细信息。现在，先实现弹出一个toast消息。

你应该已经注意到了，尽管RecyclerView功能强大，但它实际上只专注于做好本职工作。（它是我们的好榜样。）因此，要自己动手处理触摸事件。当然，如果真的需要，RecyclerView也能帮你转发触摸事件；不过大多数时候没有必要这样做。

很自然，我们想到的常用解决方案是设置OnClickListener监听器。既然列表项视图都关联着ViewHolder，就可以让ViewHolder为它监听用户触摸事件。

我们通过修改CrimeHolder类来处理用户点击事件，如代码清单8-24所示。

代码清单8-24　检测用户点击事件（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
        super(inflater.inflate(R.layout.list_item_crime, parent, false));
        itemView.setOnClickListener(this);
        ...
    }
```

```
        ...
        @Override
        public void onClick(View view) {
            Toast.makeText(getActivity(),
                    mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
                    .show();
        }
    }
```

在以上代码中，CrimeHolder类实现了OnClickListener接口；而对于itemView来说，CrimeHolder承担了接收用户点击事件的任务。

运行CriminalIntent应用。点击某个列表项，可看到弹出的toast响应消息。

8.6　深入学习：ListView 和 GridView

Android操作系统核心库包含ListView、GridView和Adapter这3个类。Android 5.0之前，创建列表项或网格项都应该优先使用这些类。

这些类的API与RecyclerView的API非常相似。ListView和GridView不关心具体的展示项，只负责展示项的滚动。Adapter负责创建列表项的所有视图。不过，使用ListView和GridView时不一定非要使用ViewHolder模式（虽然可以并且应该使用）。

过去传统的实现方式现已被RecyclerView的实现方式取代，因此不用再费力地调整ListView和GridView的工作行为了。

举例来说，ListView API不支持创建水平滚动的ListView，因此需要许多额外的定制工作。使用RecyclerView时，虽然创建定制布局和滚动行为也需要额外的工作，但RecyclerView天生支持拓展，所以使用体验还不错。

此外，RecyclerView还有支持列表项动画效果的优点。如果要让ListView和GridView支持添加和删除列表项的动画效果，实现起来既复杂又容易出错；而对于天生支持动画特效的RecyclerView来说，对付这些任务简直是小菜一碟。

口吐莲花，不如直接秀代码。例如，如果crime列表项要从位置0移动到位置5，下面这段代码就可以做到。

```
mRecyclerView.getAdapter().notifyItemMoved(0, 5);
```

8.7　深入学习：单例

Android开发实践中，经常会用到CrimeLab中使用过的单例模式。然而，单例若使用不当，会导致应用难以维护，因此它也常遭人诟病。

Android开发常用到单例的一大原因是，它们比fragment或activity活得久。例如，在设备旋转或是在fragment和activity间跳转的场景下，单例不会受到影响，而旧的fragment或activity已经不复存在了。

单例能方便地存储和控制模型对象。假设有一个比CriminalIntent更为复杂的应用，它的许多个activity和fragment会修改crime数据。某个控制单元修改了crime数据之后，怎么保证发送给其他控制单元的是最新数据呢？如果CrimeLab掌控数据对象，所有的修改都由它来处理，是不是控制数据的一致性就容易多了？而且，在控制单元间流转时，你还可以给每个crime添加ID标识，让控制单元使用ID标识从CrimeLab获取完整的crime数据。

再来谈谈单例的缺点。举个例子，虽然单例能存储数据，活得也比控制单元长，但这并不代表它能永存。在我们切换至其他应用，或逢Android回收内存时，单例连同那些实例变量也就不复存在了。结论很明显：单例无法做到持久存储。（将文件写入磁盘或是发送到Web服务器是不错的数据持久化存储方案。）

单例还不利于单元测试。例如，如果应用代码直接调用CrimeLab对象的静态方法，测试时以模拟版本的CrimeLab代替实际CrimeLab实例就不太现实。实践中，Android开发人员会使用*依赖注入*工具解决这个问题。这个工具允许以单例模式使用对象，对象也可以按需替换。

使用单例很方便，因而它很容易被滥用。在想用就用、想存就存之前，希望你能深思熟虑：数据究竟用在哪里？用在哪里能真正解决问题？

假如不慎重对待这个问题，很可能后来人在查看你的单例代码时，就像打开了一个乱糟糟的废品抽屉，里面堆满了废电池、拉链扣、旧照片，等等。它们有什么存在的意义？再强调一次：请确保有充足的理由使用单例模式存储你的共享数据！

8

若使用得当，单例就是架构优秀的Android应用中的关键部件。

8.8 挑战练习：RecyclerView ViewType

请在RecyclerView中创建两类列表项：一般性crime，以及需警方介入的crime。要完成这个挑战，你需要用到RecyclerView.Adapter的视图类别功能（view type）。在Crime对象里，再添加一个mRequiresPolice实例变量，使用它并借助getItemViewType(int)方法（developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html#getItemViewType(int)），确定该加载哪个视图到CrimeAdapter。

在onCreateViewHolder(ViewGroup, int)方法里，基于getItemViewType(int)方法返回的viewType值，需要返回不同的ViewHolder。如果是一般性crime，就仍然使用原始布局；如果是需警方介入的crime，就使用一个带联系警方按钮的新布局。

第 9 章 使用布局与组件创建用户界面

本章，我们来给RecyclerView列表项添加一些样式，借此学习更多有关布局和组件的知识。同时，我们还会重点学习使用一个叫作ConstraintLayout的新工具。至本章结束时，CrimeListFragment视图会有明显改观（图9-1），整个应用看起来更加大气漂亮。



图9-1　美观大气的CriminalIntent应用

不过，在迫不及待探索ConstraintLayout新工具之前，先来做点准备工作。你需要把图9-1中漂亮的手铐图像复制一份放入项目。浏览随书文件，找到并打开09_LayoutsAndWidgets/CriminalIntent/app/src/main/res目录，把各个版本的ic_solved.png复制到项目对应的drawable目录里。如果忘记怎么做，请温习2.6节。

9.1　使用图形布局工具

目前为止，布局都是以手动输入XML的方式创建的。本节，我们开始使用图形布局工具。

打开list_item_crime.xml布局文件，然后选择窗口底部的Design标签页。

图形布局工具界面的中间区域是布局的界面预览窗口。右边紧挨的是蓝图（blueprint）视图。蓝图和预览视图有点像，但它能显示各个组件视图的轮廓。预览让你看到视图长什么样，而从蓝图可以看出各个组件视图的大小比例。

图形布局工具界面的左边是*组件面板视图*，它包含了所有你可能用到的组件，按类别组织（如图9-2所示）。

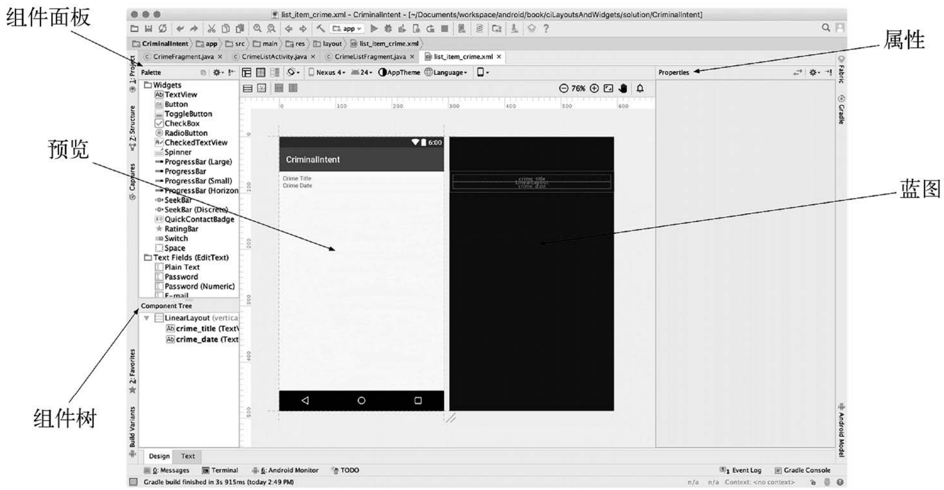


图9-2　图形布局工具中的视图

图形布局工具界面的左下是*组件树*，组件树表明组件是如何在布局中组织的。

图形布局工具界面的右边是*属性视图*。在此视图中，你可以查看并编辑组件树中已选中的组件属性。

9.2　引入 ConstraintLayout

我们可以不使用嵌套布局，而使用ConstraintLayout工具给布局添加一系列约束。可以把约束想象为橡皮筋。它会向中间拉拢分系两头的东西。例如，如图9-3所示，从ImageView视图右边到其父视图右边（ConstraintLayout自己），你可以添加一个约束。这个约束就向右拉着ImageView视图。



图9-3　右边添加了约束的ImageView

你也可以创建四个方向上的约束（左、上、右和下）。如图9-4所示，如果创建两个相反方向的约束，它们会均等地向相反的方向拉，ImageView视图就会处于正中间位置。

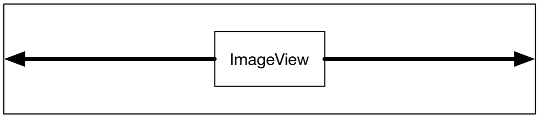


图9-4　两边都有约束的ImageView

综上所述可以得出重点：想要在ConstraintLayout里布置视图，不用再麻烦地拖来拖去了，给它们添加上约束就可以了。

位置摆放有办法了，那如何控制组件大小呢？有三个选择：让组件自己决定（使用wrap_content）；手动调整；让组件充满约束布局。

有了上述组件布置方法，只需一个ConstraintLayout，就可以布置多个布局。嵌套布局终于可以休息了。接下来，我们就一起来看看如何使用约束布置list_item_crime布局。

9.2.1　使用 ConstraintLayout

首先转换list_item_crime.xml布局，改用ConstraintLayout。如图9-5所示，在组件树窗口，右键单击根LinearLayout，然后选择Convert LinearLayout to ConstraintLayout菜单项。

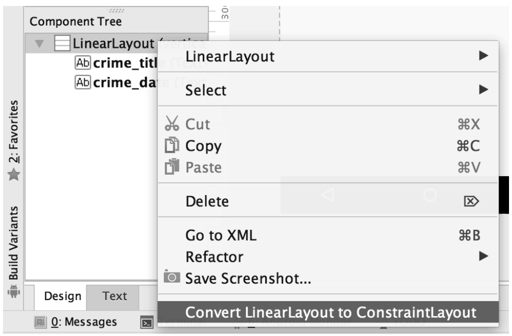


图9-5　转换根视图为ConstraintLayout

如图9-6所示，Android Studio会弹出一个窗口，让你确认如何转换。list_item_crime.xml是个简单布局，不需要深度优化。所以，接受默认值，点击OK按钮确认。

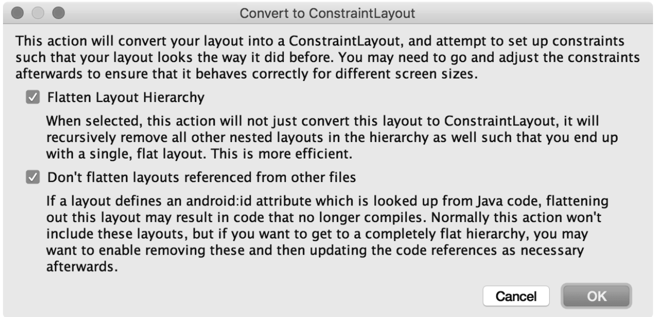


图9-6　转换默认配置

最后，如图9-7所示，你需要确认在项目里添加约束布局依赖项。和RecyclerView类似，ConstraintLayout也包含在某个库里。所以，若要使用它，要么手动添加，要么点击OK按钮确认，让Android Studio帮你添加。

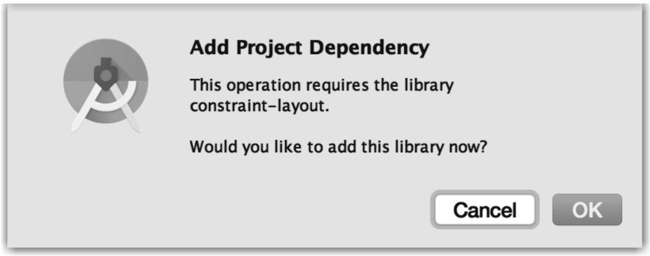


图9-7　添加ConstraintLayout依赖项

查看app/build.gradle文件，可以看到依赖项已经添加成功，如代码清单9-1所示。

代码清单9-1　内边距属性的实际应用（fragment_crime.xml）

```
dependencies {
    ...
    compile 'com.android.support.constraint:constraint-layout:1.0.0-beta4'
}
```

就这样，LinearLayout组件已经成功转换为ConstraintLayout组件。

9.2.2　约束编辑器

如图9-8所示，靠近布局预览窗口顶部的工具栏上，你可以看到一些约束编辑选项。下面看一下它们分别有什么作用。

- 显示所有约束

  显示你在预览和蓝图视图里创建的所有约束。这个控制项有时很有用，有时帮倒忙。如果你有大量的约束，点击这个控制按钮，估计你会得密集恐惧症。

- 自动连接切换开关

  启用后，在预览界面拖移视图时，约束会自动配置。Android Studio会猜测你的视图布置意图，帮你自动连接。
- 清除全部约束

  清除布局中的所有约束。稍后就会用。
- 猜测约束

  这个选项类似自动连接，Android Studio会自动帮你创建约束。任何时候，只要你向布局文件添加视图，自动连接都会被激活。

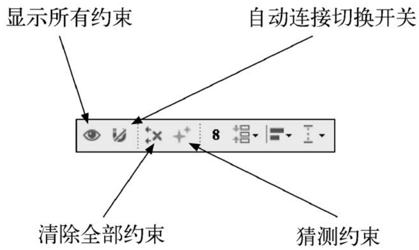


图9-8　约束编辑选项

转换list_item_crime使用`ConstraintLayout`时，根据原布局的视图布置，Android Studio已经自动添加了约束。不过，为了观察学习，我们得从头开始。

在组件树里选择`ConstraintLayout`，然后选择图9-8里的“清除全部约束”选项。你会立即看到红色警告标志，上面显示数字4。点击它看看究竟怎么回事（图9-9）。

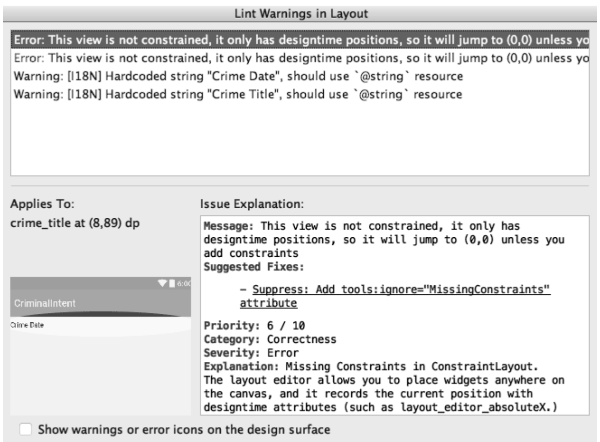


图9-9　`ConstraintLayout`警告

原来，视图没有足够的约束，`ConstraintLayout`不知道该如何布局了。`TextView`组件根本没有约束，所以它们都收到警告说，运行时可能不能出现在正确位置。

稍后，我们会添加上需要的约束修正这个问题。添加过程中，注意查看是否有警告信息，以避免运行时的异常行为。

9.2.3　腾出空间

两个TextView组件占据了整个区域，再难容下其他组件。现在，需要把它们缩小。

在组件树里，选中crime_title，然后查看右边的属性视图窗口，如图9-10所示。

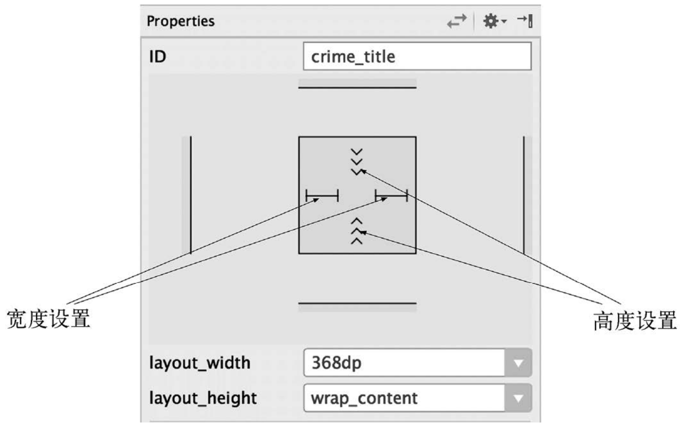


图9-10　TextView的属性

TextView水平方向和竖直方向的尺寸是分别由宽度设置和高度设置决定的。能设置的值有以下三种，如图9-11所示。每种值都对应layout_width或layout_height的一个值。

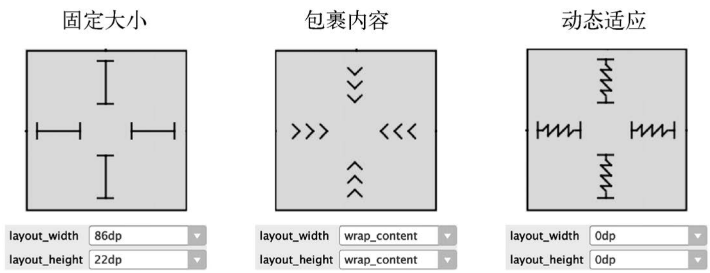


图9-11　三种视图尺寸设置

表9-1　视图尺寸设置类型

设置类型	设置值	用　　法
固定大小	Xdp	以dp为单位，为视图指定固定值（dp稍后介绍）
包裹内容	wrap_content	设置视图想要的尺寸（随内容走），也就是说，大到足够容纳内容
动态适应	0dp	允许视图缩放以满足指定约束

当前，crime_title和crime_date都设定了一个最大的固定宽度值，所以占据了整个屏幕。选中crime_title，把宽度值设为wrap_content，如果有必要，把高度值也设为wrap_content，如图9-12所示。

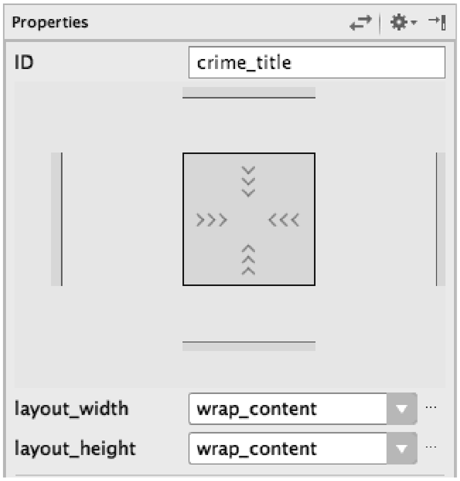


图9-12　调整crime_title的宽高值

重复上述步骤，设置crime_date的宽高值。现在这两个组件小一些了，并且还重叠在一起了，如图9-13所示。

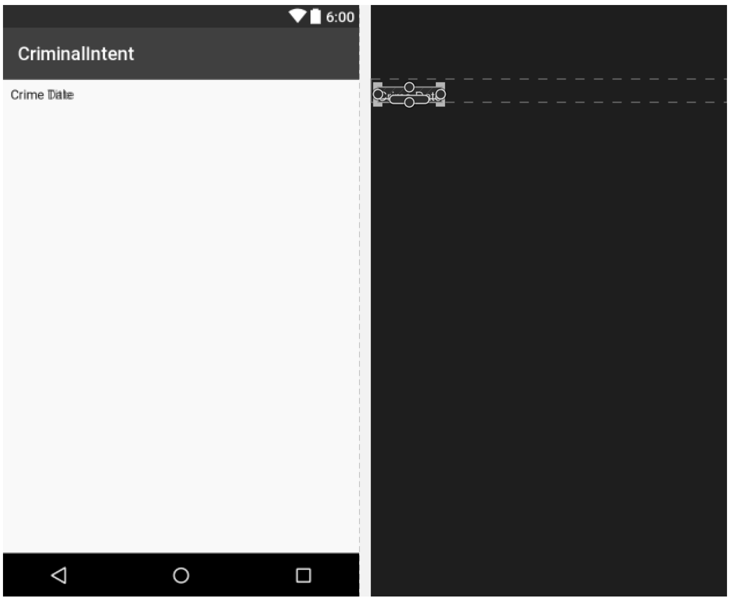


图9-13　重叠的TextView

9.2.4　添加组件

处理完两个TextView，可以向布局里添加手铐图片了。首先添加一个ImageView视图。在组件面板里找到ImageView组件（图9-14），把它拖入组件树，并作为ConstraintLayout的子组件，放在crime_date下面。



图9-14　找到ImageView组件

在随后弹出的对话框里，选择ic_solved作为ImageView组件的资源，如图9-15所示。这个图片用来表明crime已经解决。

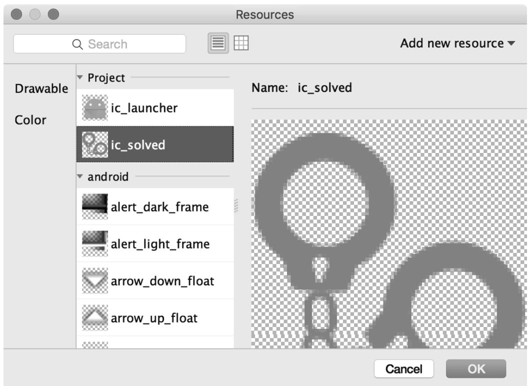


图9-15　选择ImageView组件资源

ImageView组件添加完了，但它还没有任何约束。虽然它现在有个位置，但这个位置没有任何意义。

现在为它添加约束。在组件树或者预览界面里选中ImageView组件，可看到它的四边都有圆点，如图9-16所示。这些点表示约束柄。



图9-16　ImageView组件的约束柄

按照设计构想，ImageView组件要放在视图的右边。这需要给ImageView组件的上、下和右三条边添加约束。

首先来设置ImageView组件顶部和ConstraintLayout组件顶部之间的约束。ConstraintLayout组件顶部不容易看到，但实际上它就在CriminalIntent蓝色工具栏的下面。在预览界面，拖住ImageView组件顶部的约束柄，将其拖向ConstraintLayout组件顶部。因为这两个组件顶部比较接近，所以拖的时候可能要多试几次，直到约束柄变绿，并弹出Release to Create Top Constraint这样的提示时（如图9-17所示），再松开鼠标。

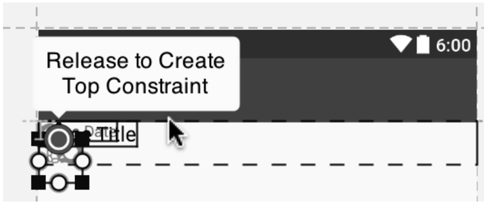


图9-17　创建顶部约束

注意，光标变为拐角形状时，不要点击，因为这会改变ImageView组件的尺寸。另外，还要小心别把约束设到TextView组件上。如果真的搞错了，就点击约束柄删掉后重做。

松开鼠标设置约束时，视图会立即就位以表明现在有了一个新约束。这就是视图在ConstraintLayout里摆放的方式——设置和删除约束。

想确认ImageView顶部和ConstraintLayout顶部是不是已经连接了约束，可在ImageView悬停鼠标，如果是，应该会出现如图9-18所示的形状。

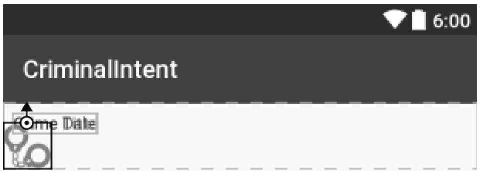


图9-18　带顶部约束的ImageView

按同样的方式，拖住ImageView组件底部的约束柄，拖到根视图的底部。同样，不要拖到TextView上。如图9-19所示，这次是向根视图的中间向下慢拖。

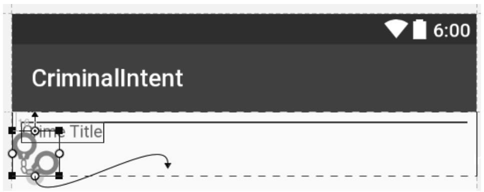


图9-19　ImageView组件约束设置进行中

最后，向根视图右边拖曳ImageView的右约束柄，设置右边约束。完成后，将鼠标悬停在ImageView组件上，确认所有的约束都正确设置，如图9-20所示。

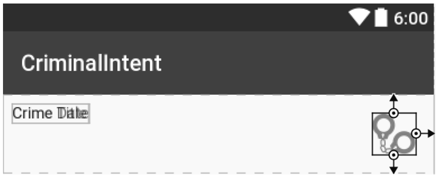


图9-20　ImageView上设置了三个约束

9.2.5　约束的 XML 形式

最终，图形约束编辑窗口的任何编辑都会体现在XML文件里。当然，如果你愿意，还是可以直接编辑原生ConstraintLayout XML。不过，显然还是使用图形约束编辑器更为方便。

将布局切换到XML文件模式，看看刚刚为ImageView创建的三个约束都向XML文件里添加了什么内容，如代码清单9-2所示。

代码清单9-2　ImageView约束的XML形式（layout/list_item_crime.xml）

```
<android.support.constraint.Constraintlayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    ...
    <imageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@drawable/ic_solved"
        android:layout_marginTop="16dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="16dp"
        android:layout_marginEnd="16dp"
        app:layout_constraintRight_toRightOf="parent"/>

</android.support.constraint.Constraintlayout>
```

以顶部约束为例，我们来审视一下它的属性设置：

```
app:layout_constraintTop_toTopOf="parent"
```

这个属性以layout_开头。凡是以layout_开头的属性都属于*布局参数*（layout parameter）。与其他属性不同的是，组件的布局参数是用来向其父组件做指示的，即用于告诉父布局如何安排自己。目前为止，我们已经见识过好几个这样的布局参数，如layout_width和layout_height。

约束的名字是constraintTop。这表示它是ImageView的顶部约束。

最后，属性以toTopof="parent"结束，这表明，约束是连接到父组件（ConstraintLayout）顶部的。

好了，一口气看了这么多，歇一歇。不过，事情还没完，又该回到图形约束编辑窗口了。

9.2.6　编辑属性

现在，ImageView组件已经摆放正确了。接下来的任务是布置和调整标题TextView组件。

首先，在组件树里选中crime_date，把它拖曳到别处，如图9-21所示。注意，在预览界面，拖到别处看上去是换了位置，但应用运行时，你依然看不到这种位置变化。不过，使用约束调整位置就可以。

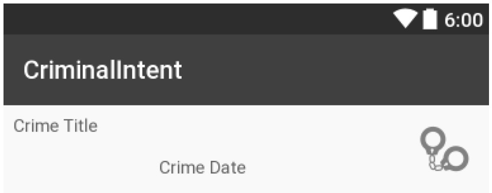


图9-21　把crime_date拖到别处

现在，在组件树里选中crime_title视图。它的目标位置是布局的左上角，ImageView组件的左边。这需要添加以下三个约束：

- 从crime_title视图的左边到其父组件的左边，带16dp的边距；
- 从crime_title视图的顶部到其父组件的顶部，带16dp的边距；
- 从crime_title视图的右边到ImageView组件的左边，带8dp的边距。

创建上述约束。（定位、拖曳需要耐心和技巧，不行就多用Command+Z或Ctrl+Z）撤销快捷键反复试几次。）

确认约束都添加完成，如图9-22所示（选中组件就能看到）。

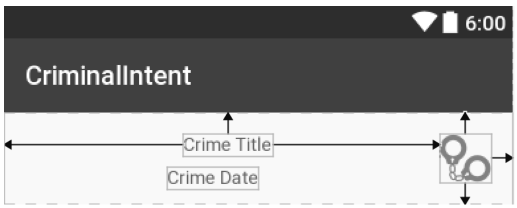


图9-22　TextView视图的约束

注意，点击TextView时，会看到它有一块椭圆区域，而ImageView没有。这是TextView用来对齐文字的特有约束锚点。不过，本章用不到，了解就行。

搞定了crime_title的约束，现在设置视图尺寸。视图宽度设置为动态适应（0dp），这样标

题文字就可以占满约束间的空间。视图高度设置为wrap_content，以便刚好显示出crime标题。最后，确认设置结果如图9-23所示。

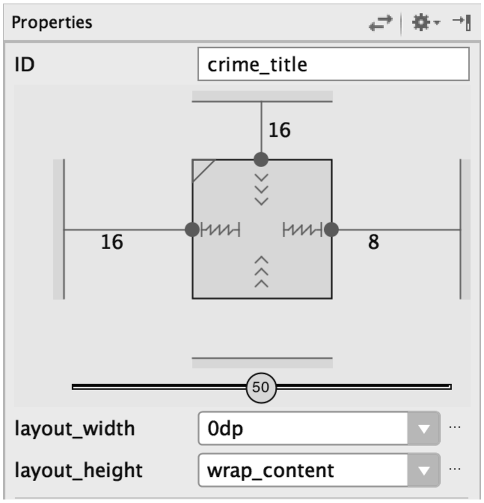


图9-23 crime_title视图设置

接下来处理crime_date视图。在组件树里选中它，为其添加以下三个约束：

- 从crime_date视图的左边到其父组件的左边，带16dp的边距；
- 从crime_date视图的顶部到crime_title视图的底部，带8dp的边距；
- 从crime_date视图的右边到ImageView组件的左边，带8dp的边距。

完成约束设置后，开始设置视图尺寸。和crime_title视图一样，设置视图宽度为动态适应（0dp），高度为wrap_content。最后，确认设置结果如图9-24所示。

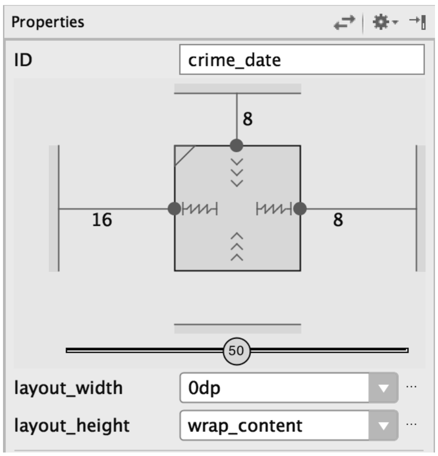


图9-24 crime_date视图设置

现在，查看预览，应该能看到类似图9-1的显示效果。

运行CriminalIntent应用，确认三个组件在RecyclerView视图都显示得上下齐整、疏落有致，如图9-25所示。

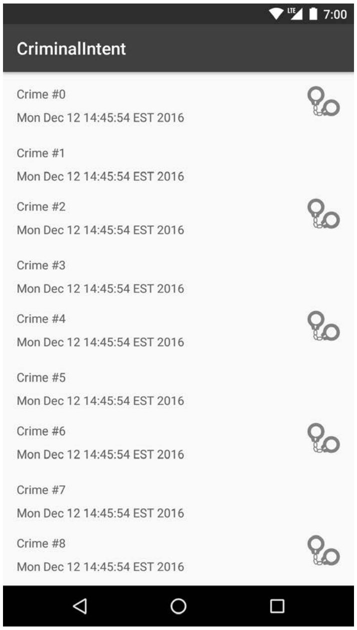


图9-25　图形布局工具中的视图

9.2.7　动态设置列表项

当前，应用运行时，每行都显示了手铐图片。这和实际不符，需要修改ImageView组件解决。

首先，更新ImageView组件的ID（组件添加时已设置了默认名）。在组件树里选中它，然后在视图属性窗口将ID修改为crime_solved，如图9-26所示。Android Studio会询问是否更新所有用到该ID的地方，点Yes确认。

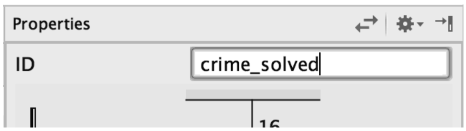


图9-26　更新ImageView组件的ID

更新完ID，代码也要做对应更新。打开CrimeListFragment.java文件，在CrimeHolder类中，添加一个ImageView实例变量。然后，根据crime记录的解决状态，控制图片的显示，如代码清单9-3所示。

代码清单9-3　控制图片显示（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClicklistener {
    ...
    private TextView mDateTextView;
    private ImageView mSolvedImageView;

    public CrimeHolder(layoutinflater inflater, ViewGroup parent) {
        super(inflater.inflate(R.layout.list_item_crime, parent, false));
        itemView.setOnClicklistener(this);

        mTitleTextView = (TextView) itemView.findViewByid(R.id.crime_title);
        mDateTextView = (TextView) itemView.findViewByid(R.id.crime_date);
        mSolvedImageView = (ImageView) itemView.findViewById(R.id.crime_solved);
    }

    public void bind(Crime crime) {
        mCrime = crime;
        mTitleTextView.setText(mCrime.getTitle());
        mDateTextView.setText(mCrime.getDate().toString());
        mSolvedImageView.setVisibility(crime.isSolved() ? View.VISIBLE : View.GONE);
    }
    ...
}
```

运行CriminalIntent应用。确认手铐图片能按问题解决情况正确显示。

9.3　深入学习布局属性

本节，我们再来微调一下list_item_crime.xml布局设计，同时解答一些可能令人困扰的组件与属性相关问题。

回到list_item_crime.xml布局图形设计界面。选中criem_title，在属性视图窗口，我们来调整一些属性。

点击textAppearance旁边的箭头，显示出各种显示文字和字体属性。修改textColor属性值为@android:color/black，如图9-27所示。



图9-27　更新标题颜色

然后，再设置textSize属性值为18sp。运行应用。可以看到，整个界面的显示效果非常好。

9.3.1　dp、sp 以及屏幕像素密度

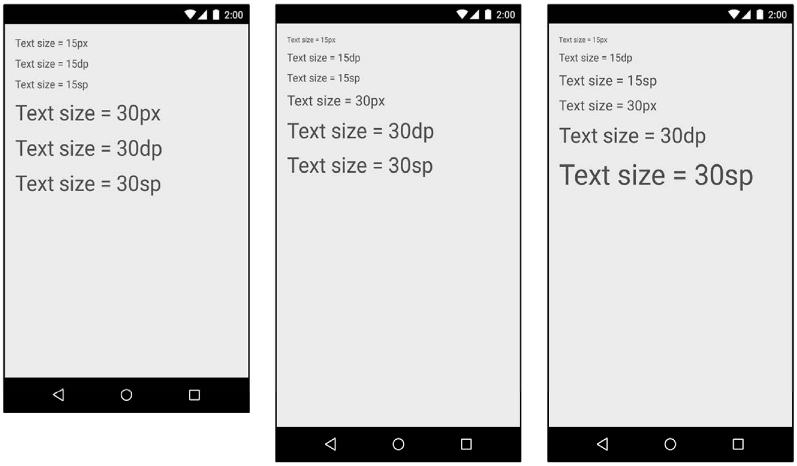
在list_item_crime.xml文件中，我们以sp和dp为单位指定了属性值。下面我们来具体学习一下。

有时需要为视图属性指定大小尺寸值（通常以像素为单位，有时也用点、毫米或英寸）。一

些常见的属性包括文字大小（text size）、边距（margin）以及内边距（padding）。文字大小指定设备上显示的文字像素高度；边距指定视图组件间的距离；内边距指定视图外边框与其内容间的距离。

在2.6节中，我们在各个带屏幕密度修饰的drawable（如`drawable-xhdpi`）下准备了对应的图片文件，Android会用它们自动适配不同像素密度的屏幕。那么问题来了，假如图片能自动适配，但边距无法缩放适配，或者用户配置了大于默认值的文字大小，会发生什么情况呢？

为解决这些问题，Android提供了与密度无关的尺寸单位。运用这种单位，可在不同屏幕像素密度的设备上获得同样的尺寸。无需麻烦的转换计算，应用运行时，Android会自动将这种单位转换成像素单位，如图9-28所示。



MDPI　　HDPI　　HDPI, 大字体

图9-28　使用与密度无关的尺寸单位时，`TextView`的显示效果

- px
  英文pixel的缩写，即像素。无论屏幕密度多少，一个像素单位对应一个屏幕像素单位。不推荐使用px，因为它不会根据屏幕密度自动缩放。
- dp（或dip）
  英文density-independent pixel的缩写，意为密度无关像素。在设置边距、内边距或任何不打算按像素值指定尺寸的情况下，通常都使用dp这种单位。如果屏幕密度较高，密度无关像素会相应扩展至整个屏幕。1dp在设备屏幕上总是等于1/160英寸。使用dp的好处是，无论屏幕密度如何，总能获得同样的尺寸。
- sp
  英文scale-independent pixel的缩写，意为缩放无关像素。它是一种与密度无关的像素，这种像素会受用户字体偏好设置的影响。通常使用sp来设置屏幕上的字体大小。

- pt、mm、in

  类似于dp的缩放单位，允许以点（1/72英寸）、毫米或英寸为单位指定用户界面尺寸。但在实际开发中不建议使用这些单位，因为并非所有设备都能按照这些单位进行正确的尺寸缩放配置。

在本书及实际开发中，通常只会用到dp和sp这两种单位。Android会在运行时自动将它们的值转换为像素单位。

9.3.2　边距与内边距

在GeoQuiz和CriminalIntent这两个应用中，我们给组件设置过边距与内边距属性。开发新手有时分不清这两个属性。既然你已明白什么是布局参数，那么二者的区别也就显而易见了。

边距属性是布局参数，决定了组件间的距离。由于组件对外界一无所知，因此边距必须由该组件的父组件负责。

内边距不是布局参数。属性`android:padding`告诉组件：在绘制组件自身时，要比所含内容大多少。举例说明：在不改变文字大小的情况下，想把日期按钮变大一些，如图9-29所示。



图9-29　实话实说，我喜欢大按钮

可将下面的属性添加给`Button`，如代码清单9-4所示。

代码清单9-4　内边距属性的实际应用（fragment_crime.xml）

```
<Button android:id="@+id/crime_date"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
```

```
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:padding="80dp" />
```

大按钮很方便，但很可惜，继续学习前，还是应该删除这个属性。

9.3.3　样式、主题及主题属性

样式（style）是XML资源文件，含有用来描述组件行为和外观的属性定义。例如，使用下列样式配置组件，就能显示比正常大小更大的文字：

```
<style name="BigTextStyle">
  <item name="android:textSize">20sp</item>
  <item name="android:padding">3dp</item>
</style>
```

你可以创建自己的样式文件（第22章会这样做）。具体做法是将属性定义添加并保存在res/values/目录下的样式文件中，然后在布局文件中以`@style/my_own_style`（样式文件名）的形式引用。

再来看看fragment_crime.xml文件中的两个`TextView`组件。每个组件都有一个引用Android自带样式文件的`style`属性。该预定义样式来自于应用的*主题*，能让屏幕上的`TextView`组件看起来是以列表样式分隔开的。*主题*是各种样式的集合。从结构上来说，主题本身也是一种样式资源，只不过它的属性指向了其他样式资源。

Android自带了一些供应用使用的平台主题。例如，在创建CriminalIntent应用时，向导就设置了默认主题（是在manifest文件的`application`标签下引用的）。

使用*主题属性引用*，可将预定义的应用主题样式添加给指定组件。在fragment_crime.xml文件中，样式属性值`?android:listSeparatorTextViewStyle`的使用就是这样一个例子。

使用主题属性引用，就是告诉Android运行资源管理器："在应用主题里找到名为`listSeparatorTextViewStyle`的属性。该属性指向其他样式资源，请将其资源的值放在这里。"

所有Android主题都包括名为`listSeparatorTextViewStyle`的属性。不过，基于特定主题的整体风格，它们的定义稍有不同。使用主题属性引用，可以确保`TextView`组件在应用中拥有正确一致的显示风格。

你还会在第22章学习到更多有关样式及主题的使用知识。

9.3.4　Android 应用的设计原则

注意看边距属性，Android Studio默认使用的值是16dp或8dp。设定这两种值遵循了Android的material design原则。访问developer.android.com/design/index.html，可看到所有的Android设计规范。

开发Android应用都应严格遵循这些设计原则。不过，这些设计原则严重依赖于SDK较新版本的功能，旧版本设备往往无法获得或实现这些功能。不过有些设计可借助AppCompat库实现，详见第13章。

9.4 图形布局工具使用小结

图形布局工具非常有用，有了ConstraintLayout更是如虎添翼。然而，人各有所爱，很多开发人员更喜欢简单直白的XML。

别犯选择困难症啦！图形布局工具也好，写XML也好，反正可以随时切换，怎么方便就怎么来吧！在后续章节中，如果需要创建布局，我们只会提供布局结构示意图，具体怎么实现，你自己定。

9.5 挑战练习：日期格式化

与其说Date对象是普通日期，不如说是时间戳。调用Date对象的toString()方法，就能得到一个时间戳。所以，RecyclerView视图上显示的就是它。时间戳虽然凑合能用，但如果能显示人们习惯看到的日期应该会更好，如“Jul 22, 2016”。要实现此目标，可使用android.text.format.DateFormat类实例。具体怎么用，请查阅Android文档库中有关该类的说明。

使用DateFormat类中的方法，可获得常见格式的日期；也可以自己定制字符串格式。最后，再来一个更有挑战的练习：创建一个包含星期的字符串格式，如“Friday, Jul 22, 2016”。

第 10 章 使用fragment argument

本章，我们将关联CriminalIntent应用的列表与明细部分。用户点击某个crime列表项时，会创建一个托管CrimeFragment的CrimeActivity，以展现Crime实例的明细信息。

在GeoQuiz应用里，我们已实现从activity（QuizActivity）中启动activity（CheatActivity）。在CriminalIntent应用里，我们要实现从fragment中启动CrimeActivity。准确地说，是从CrimeListFragment中启动CrimeActivity实例，如图10-1所示。

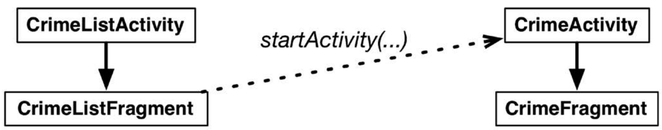


图10-1　从CrimeListActivity中启动CrimeActivity

10.1　从 fragment 中启动 activity

从fragment中启动activity类似于从activity中启动activity。我们调用Fragment.startActivity(Intent)方法，由它在后台再调用对应的Activity方法。

在CrimeListFragment的CrimeHolder类里，用启动CrimeActivity实例的代码，替换toast消息处理代码，如代码清单10-1所示。

代码清单10-1　启动CrimeActivity（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        Toast.makeText(getActivity(),
                mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
                .show();
        Intent intent = new Intent(getActivity(), CrimeActivity.class);
        startActivity(intent);
    }
}
```

指定要启动的activity为CrimeActivity，CrimeListFragment创建了一个显式intent。至于Intent构造方法需要的Context对象，CrimeListFragment是通过使用getActivity()方法传入它的托管activity来满足的。

运行CriminalIntent应用。点击任意列表项，托管CrimeFragment的新CrimeActivity随即出现在屏幕上，如图10-2所示。

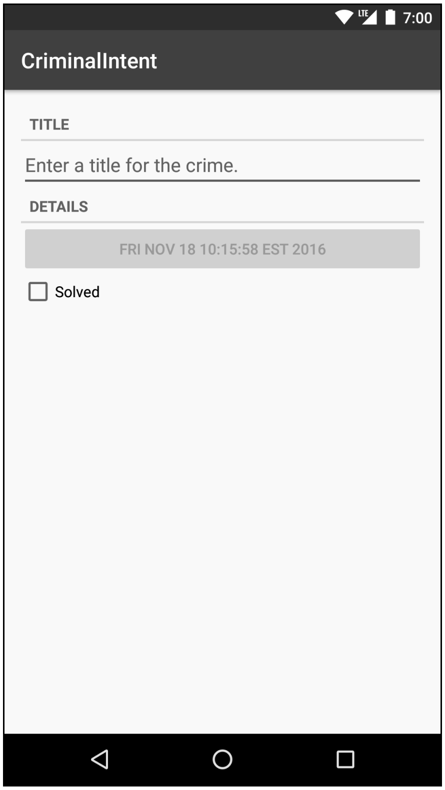


图10-2　空白的CrimeFragment

由于不知道该显示哪个Crime对象，因此CrimeFragment没有显示出具体的Crime信息。

10.1.1　附加 extra 信息

启动CrimeActivity时，传递附加到Intent extra上的crime ID，CrimeFragment就能知道该显示哪个Crime。

这需要在CrimeActivity中新增newIntent方法，如代码清单10-2所示。

代码清单10-2　创建newIntent方法（CrimeActivity.java）

```
public class CrimeActivity extends SingleFragmentActivity {

    public static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";

    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimeActivity.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }
    ...
}
```

创建了显式intent后，调用putExtra(...)方法，传入匹配crimeId的字符串键与键值。这里，由于UUID是Serializable对象，因此我们需要调用putExtra(String, Serializable)方法。

更新CrimeHolder，使用newIntent方法，如代码清单10-3所示。

代码清单10-3　传递Crime实例（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getActivity(), CrimeActivity.class);
        Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
        startActivity(intent);
    }
}
```

10.1.2　获取 extra 信息

crime ID现已安全存储到CrimeActivity的intent中。然而，要获取和使用extra信息的是CrimeFragment类。

fragment有两种方式获取intent中的数据：一种简单直接，另一种复杂但比较灵活（涉及fragment argument的概念）。

我们首先来看简单的方式：CrimeFragment直接使用getActivity()方法获取CrimeActivity的intent。回到CrimeFragment.java文件，取到CrimeActivity的intent内的extra信息，再用它获取Crime对象，如代码清单10-4所示。

代码清单10-4　获取extra数据并取得Crime对象（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    ...
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
```

```
        UUID crimeId = (UUID) getActivity().getIntent()
                .getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    }
    ...
}
```

在代码清单10-4中，除了getActivity()方法的调用，获取extra数据的实现代码与activity里获取extra数据的代码一样。getIntent()方法返回用来启动CrimeActivity的Intent，然后调用Intent的getSerializableExtra(String)方法获取UUID并存入变量中。

取得Crime的ID后，再用它从CrimeLab单例中调取Crime对象。

10.1.3　使用 Crime 数据更新 CrimeFragment 视图

既然获取到了Crime对象，CrimeFragment视图便可显示该Crime对象的数据了。参照代码清单10-5，更新onCreateView(...)方法，显示Crime对象的标题及解决状态。（显示日期的代码早已就绪。）

代码清单10-5　更新视图对象（CrimeFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mTitleField = (EditText)v.findViewById(R.id.crime_title);
    mTitleField.setText(mCrime.getTitle());
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
    });
    ...
    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setChecked(mCrime.isSolved());
    mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        ...
    });
    ...
    return v;
}
```

运行CriminalIntent应用。选中Crime #4，查看显示了正确信息的CrimeFragment实例，如图10-3所示。



图10-3　Crime #4列表项的明细内容

10.1.4　直接获取 extra 信息的缺点

只需几行简单的代码，就可让fragment直接从托管activity的intent中获取信息。然而，这种方式破坏了fragment的封装。`CrimeFragment`不再是可复用的构建单元，因为它现在由某个特定的activity托管着，该特定activity的`Intent`又定义了名为`com.bignerdranch.android.criminalintent.crime_id`的extra。

就`CrimeFragment`类来说，这看起来合情合理；但这也意味着，按照当前的编码实现，`CrimeFragment`便再也无法用于任何其他的activity了。

一个比较好的做法是，将crime ID存储在属于`CrimeFragment`的某个地方，而不是保存在`CrimeActivity`的私有空间里。这样，无需依赖`CrimeActivity`的intent内的extra，`CrimeFragment`就能获取自己所需的extra数据。属于fragment的“某个地方”实际就是它的argument bundle。

10.2　fragment argument

每个fragment实例都可附带一个`Bundle`对象。该bundle包含键–值对，我们可以像附加extra到`Activity`的intent中那样使用它们。一个键–值对即一个argument。

要创建fragment argument，首先需创建`Bundle`对象。然后，使用`Bundle`限定类型的put方法（类似于`Intent`的方法），将argument添加到bundle中（代码如下所示）。

```
Bundle args = new Bundle();
args.putSerializable(ARG_MY_OBJECT, myObject);
args.putInt(ARG_MY_INT, myInt);
args.putCharSequence(ARG_MY_STRING, myString);
```

10.2.1　附加 argument 给 fragment

要附加argument bundle给fragment，需调用Fragment.setArguments(Bundle)方法。而且，还必须在fragment创建后、添加给activity前完成。

为满足以上要求，Android开发人员采取的习惯做法是：添加名为newInstance()的静态方法给Fragment类。使用该方法，完成fragment实例及Dundle对象的创建，然后将argument放入bundle中，最后再附加给fragment。

托管activity需要fragment实例时，转而调用newInstance()方法，而非直接调用其构造方法。而且，为满足fragment创建argument的要求，activity可给newInstance()方法传入任何需要的参数。

在CrimeFragment类中，编写可以接受UUID参数的newInstance(UUID)方法，创建argument bundle和fragment实例，然后附加argument给fragment，如代码清单10-6所示。

代码清单10-6　编写newInstance(UUID)方法（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";

    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckbox;

    public static CrimeFragment newInstance(UUID crimeId) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_CRIME_ID, crimeId);

        CrimeFragment fragment = new CrimeFragment();
        fragment.setArguments(args);
        return fragment;
    }
    ...
}
```

现在，需创建CrimeFragment时，CrimeActivity应调用CrimeFragment.newInstance(UUID)方法，并传入从它的extra中获取的UUID参数值。回到CrimeActivity类中，在createFragment()方法里，从CrimeActivity的intent中获取extra数据，并传入CrimeFragment.newInstance(UUID)方法，如代码清单10-7所示。

既然其他类不会用到EXTRA_CRIME_ID，可以将其改为私有。

代码清单10-7　使用newInstance(UUID)方法（CrimeActivity.java）

```
public class CrimeActivity extends SingleFragmentActivity {

    public private static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";
    ...
    @Override
```

```
    protected Fragment createFragment() {
        return new CrimeFragment();
        UUID crimeId = (UUID) getIntent()
                .getSerializableExtra(EXTRA_CRIME_ID);
        return CrimeFragment.newInstance(crimeId);
    }
}
```

注意，activity和fragment不需要也无法同时相互保持独立。CrimeActivity必须了解CrimeFragment的内部细节，比如知道它内部有个newInstance(UUID)方法。这很正常。托管activity应该知道这些细节，以便托管fragment；但fragment不一定需要知道其托管activity的细节问题，至少在需要保持fragment通用独立的时候如此。

10.2.2　获取 argument

fragment需要获取它的argument时，会先调用Fragment类的getArguments()方法，再调用Bundle限定类型的get方法，如getSerializable(...)方法。

现在回到CrimeFragment.onCreate(...)方法中，改为从fragment的argument中获取UUID，如代码清单10-8所示。

代码清单10-8　从argument中获取crime ID（CrimeFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getActivity().getIntent()
            .getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);

}
```

运行CriminalIntent应用。虽然运行结果一样，但有理由高兴高兴，因为我们不仅让CrimeFragment类变得通用了，还为下一章实现更为复杂的列表项导航打下了良好基础。

10.3　刷新显示列表项

运行CriminalIntent应用，点击某个列表项，然后修改对应的Crime明细信息。这些修改数据已保存至模型层，但返回列表后，RecyclerView视图并没有刷新。下面就来处理这个问题。

模型层保存的数据若有变化（或可能有变），应通知RecyclerView的Adapter，以便其及时获取最新数据并刷新显示列表项。在恰当的时机，与系统的ActivityManager回退栈协同运作，可实现列表项的刷新。

CrimeListFragment启动CrimeActivity实例后，CrimeActivity被置于回退栈顶。这导致原先处于栈顶的CrimeListActivity实例被暂停并停止。

用户点击后退键返回到列表项界面，CrimeActivity随即弹出栈外并被销毁。此时，Crime-

ListActivity立即重新启动并恢复运行。应用的回退栈如图10-4所示。



图10-4　CriminalIntent应用的回退栈

CrimeListActivity恢复运行后，操作系统会发出调用onResume()生命周期方法的指令。CrimeListActivity接到指令后，它的FragmentManager会调用当前被activity托管的fragment的onResume()方法。这里的fragment就是指CrimeListFragment。

在CrimeListFragment中，覆盖onResume()方法，触发调用updateUI()方法刷新显示列表项，如代码清单10-9所示。如果已配置好CrimeAdapter，就调用notifyDataSetChanged()方法来修改updateUI()方法。

代码清单10-9　在onResume()方法中刷新列表项（CrimeListFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    ...
}

@Override
public void onResume() {
    super.onResume();
    updateUI();
}

private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
```

```
        } else {
            mAdapter.notifyDataSetChanged();
        }
    }
```

为什么选择覆盖onResume()方法来刷新显示列表项，而不用onStart()方法呢？当有其他activity位于你的activity之前时，你无法确定自己的activity是否会被停止。如果前面的activity是透明的，你的activity可能会被暂停。对于此场景下暂停的activity，onStart()方法中的更新代码是不会起作用的。一般来说，要保证fragment视图得到刷新，在onResume()方法内更新代码是最安全的选择。

运行CriminalIntent应用。选择某个crime项并修改其明细内容，然后返回到列表项界面。可以看到，列表项已经刷新。

经过前两章的开发，CriminalIntent应用已获得大幅改进。现在，再来看看升级后的应用对象图解，如图10-5所示。

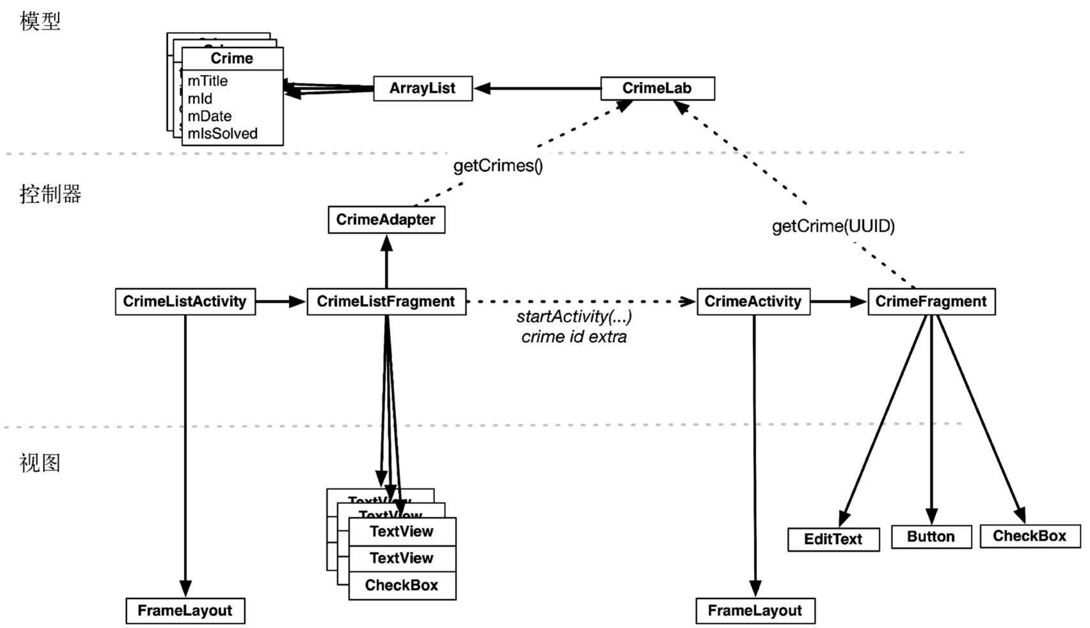


图10-5　应用对象图解更新版

10.4　通过 fragment 获取返回结果

如需从已启动的activity获取返回结果，可使用与GeoQuiz应用中类似的实现代码。具体的代码调整就是：不调用Activity的startActivityForResult(...)方法，转而调用Fragment.

startActivityForResult(...)方法；不覆盖Activity.onActivityResult(...)方法，转而覆盖Fragment.onActivityResult(...)方法。

```
public class CrimeListFragment extends Fragment {

    private static final int REQUEST_CRIME = 1;
    ...
    private class CrimeHolder extends RecyclerView.ViewHolder
            implements View.OnClickListener {
        ...
        @Override
        public void onClick(View view) {
            Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
            startActivityForResult(intent, REQUEST_CRIME);
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == REQUEST_CRIME) {
            // Handle result
        }
    }
    ...
}
```

除将返回结果从托管activity传递给fragment的额外实现代码之外，Fragment.startActivity-ForResult(Intent,int)方法类似于Activity的同名方法。

从fragment中返回结果的处理稍有不同。fragment能够从activity中接收返回结果，但其自身无法持有返回结果。只有activity拥有返回结果。因此，尽管Fragment有自己的startActivityFor-Result(...)方法和onActivityResult(...)方法，但没有setResult(...)方法。

相反，应让托管activity返回结果值，具体代码如下。

```
public class CrimeFragment extends Fragment {
    ...
    public void returnResult() {
        getActivity().setResult(Activity.RESULT_OK, null);
    }
}
```

10.5 深入学习：为何要用 fragment argument

fragment argument的使用有点复杂。为什么不直接在CrimeFragment里创建一个实例变量呢？

创建实例变量的方式并不可靠。这是因为，在操作系统重建fragment时（设备配置发生改变）用户暂时离开当前应用（操作系统按需回收内存），任何实例变量都将不复存在。尤其是内存不够，操作系统强制杀掉应用的情况，可以说是无人能挡。

因此，可以说，fragment argument就是为应对上述场景而生。

还有另一个方法应对上述场景，那就是使用实例状态保存机制。具体来说，就是将crime ID赋值给实例变量，然后在onSaveInstanceState(Bundle)方法中保存下来。要用时，从onCreate(Bundle)方法中的Bundle中取回。

然而，这种解决方案的维护成本高。举例来说，如果你在若干年后要修改fragment代码以添加其他argument，很可能会忘记在onSaveInstanceState(Bundle)方法里保存新增的argument。

Android开发人员更喜欢fragment argument这个解决方案，因为这种方式很清楚直白。若干年后，再回头修改老代码时，一眼就能知道，crime ID是以argument保存和传递使用的。即使要新增argument，也会记得使用argument bundle保存它。

10.6　挑战练习：实现高效的 RecyclerView 刷新

Adapter的notifyDataSetChanged方法会通知RecyclerView刷新全部的可见列表项。

在CriminalIntent应用里，这个方法不够高效。这是因为，返回CrimeListFragment时，最多只有一个Crime实例会发生变化。

只需要刷新列表项中的单个crime项的话，应该使用RecyclerView.Adapter的notifyItemChanged(int)方法。修改代码调用这个方法很简单，但如何定位并刷新具体位置的列表项呢？这是一个挑战！

10.7　挑战练习：优化 CrimeLab 的表现

CrimeLab的getCrime(UUID)方法没毛病，但匹配要找的crime ID这个过程还可以再优化。请优化匹配逻辑，不过重构代码时，不要搞坏了CriminalIntent应用。

第 11 章 使用ViewPager

本章，我们将创建一个新的activity，用来托管CrimeFragment。新建activity的布局将由一个ViewPager实例组成。为UI添加ViewPager后，用户可左右滑动屏幕，切换查看不同列表项的明细页面，如图11-1所示。



图11-1　滑屏切换显示Crime明细内容

图11-2为升级后的CriminalIntent应用对象图解。可以看到，名为CrimePagerActivity的新建activity取代了CrimeActivity。其布局由一个ViewPager组成。

如图11-2所示，无需改变CriminalIntent应用的其他部分，我们只要创建虚线框中的对象就能实现滑屏切换Crime明细页面。特别要说的是，经过上一章的封装，CrimeFragment类既通用又独立，因此这里就不用管它了。

接下来，我们需要完成以下任务：

- 创建CrimePagerActivity类；
- 定义包含ViewPager的视图层级结构；
- 在CrimePagerActivity类中关联使用ViewPager及其Adapter；

- 修改CrimeHolder.onClick(...)方法，转而启动CrimePagerActivity。

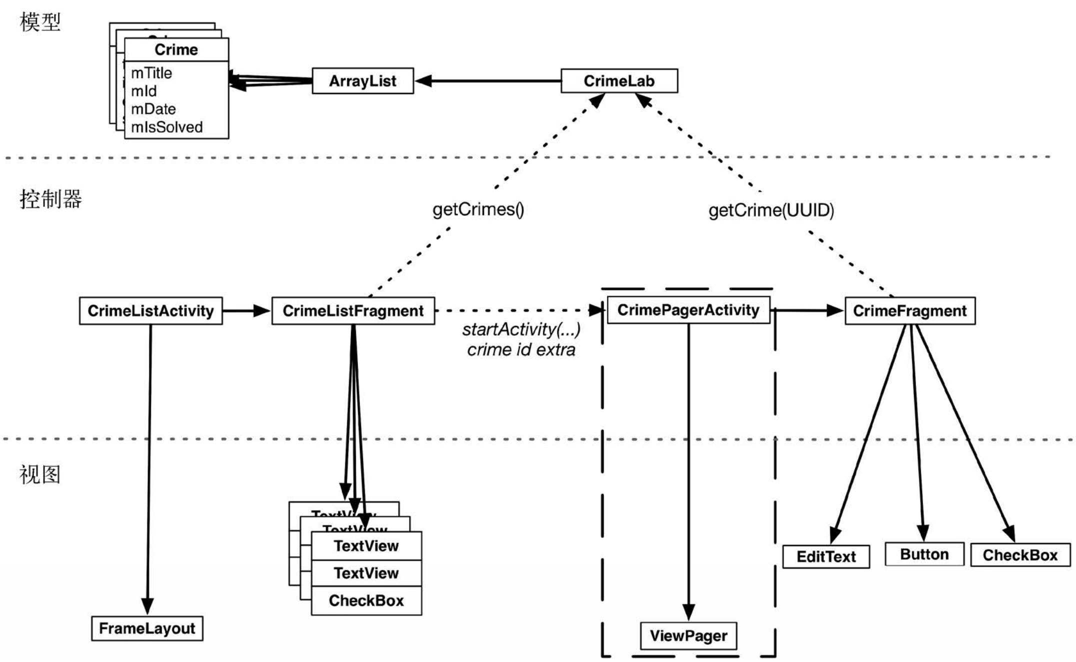


图11-2　CrimePagerActivity的对象图解

11.1　创建 CrimePagerActivity

CrimePagerActivity是AppCompatActivity的子类。在CriminalIntent应用中，其任务是创建并管理ViewPager。

以AppCompatActivity为超类，创建CrimePagerActivity新类并为其配置视图，如代码清单11-1所示。

代码清单11-1　创建ViewPager（CrimePagerActivity.java）

```
public class CrimePagerActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime_pager);
    }
}
```

参照图11-3，以ViewPager为根视图，在res/layout/目录下创建名为activity_crime_pager的布局文件。注意，必须使用ViewPager的完整包名（android.support.v4.view.ViewPager）。

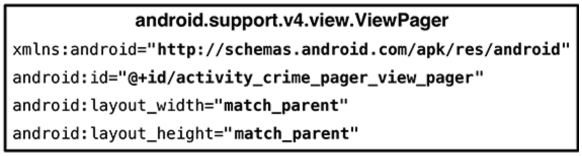


图11-3　CrimePagerActivity的ViewPager布局（activity_crime_pager.xml）

因为ViewPager类来自于支持库，所以添加到布局时包名要完整。与Fragment类不同，ViewPager仅存在于支持库。而且，在SDK的后续版本中，Google也没有在标准库中实现ViewPager类。

11.1.1　ViewPager 与 PagerAdapter

ViewPager在某种程度上类似于RecyclerView。RecyclerView需借助于Adapter提供视图。同样，ViewPager需要PagerAdapter的支持。

不过，相较于RecyclerView与Adapter间的协同工作，ViewPager与PagerAdapter间的配合要复杂得多。好在Google提供了PagerAdapter的子类FragmentStatePagerAdapter，它能协助处理许多细节问题。

FragmentStatePagerAdapter化繁为简，提供了两个有用的方法：getCount()和getItem(int)。调用getItem(int)方法，获取并显示crime数组中指定位置的Crime时，它会返回配置过的CrimeFragment来显示指定的Crime。

在CrimePagerActivity中，设置ViewPager的pager adapter，并实现它的getCount()方法和getItem(int)方法，如代码清单11-2所示。

代码清单11-2　设置pager adapter（CrimePagerActivity.java）

```
public class CrimePagerActivity extends AppCompatActivity {

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime_pager);

        mViewPager = (ViewPager) findViewById(R.id.crime_view_pager);

        mCrimes = CrimeLab.get(this).getCrimes();
        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {

            @Override
            public Fragment getItem(int position) {
                Crime crime = mCrimes.get(position);
                return CrimeFragment.newInstance(crime.getId());
```

```
            }

            @Override
            public int getCount() {
                return mCrimes.size();
            }
        });
    }
}
```

下面来逐行解读新增代码。在activity视图中找到ViewPager后，我们从CrimeLab中（crime的List）获取数据集，然后获取activity的FragmentManager实例。

接下来，设置adapter为FragmentStatePagerAdapter的一个匿名实例。创建Fragment-StatePagerAdapter实例需要FragmentManager。如前所述，FragmentStatePagerAdapter是我们的代理，负责管理与ViewPager的对话并协同工作。代理需首先将getItem(int)方法返回的fragment添加给activity，然后才能使用fragment完成自己的工作。这也就是创建代理实例时，需要FragmentManager的原因。

（代理究竟做了哪些工作呢？简单来说，就是将返回的fragment添加给托管activity，并帮助ViewPager找到fragment的视图并一一对应，详见11.3节。）

pager adapter的两个方法简单直接。getCount()方法返回数组列表中包含的列表项数目。getItem(int)方法才是神奇所在。它首先获取数据集中指定位置的Crime实例，然后利用该Crime实例的ID创建并返回一个经过有效配置的CrimeFragment。

11.1.2 整合并配置使用 CrimePagerActivity

现在，弃用CrimeActivity，我们来配置使用CrimePagerActivity。

首先新增newIntent方法和crime ID的extra常量，如代码清单11-3所示。

代码清单11-3 创建newIntent方法（CrimePagerActivity.java）

```
public class CrimePagerActivity extends AppCompatActivity {
    private static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimePagerActivity.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime_pager);
```

```
        UUID crimeId = (UUID) getIntent()
                .getSerializableExtra(EXTRA_CRIME_ID);
        ...
    }
}
```

然后需要修改CrimeListFragment，使得用户单击某个列表项时，CrimeListFragment启动的是CrimePagerActivity实例。

回到CrimeListFragment.java文件，修改CrimeHolder.onClick(View)方法以启动CrimePagerActivity，如代码清单11-4所示。

代码清单11-4　配置启动CrimePagerActivity（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    @Override
    public void onClick(View view) {
        Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
        Intent intent = CrimePagerActivity.newIntent(getActivity(), mCrime.getId());
        startActivity(intent);
    }
}
```

最后，要让操作系统启动CrimePagerActivity，还要在manifest配置文件中添加它，如代码清单11-5所示。打开AndroidManifest.xml，添加CrimePagerActivity声明，同时删除不再使用的CrimeActivity声明；也可以直接将CrimeActivity重命名为CrimePagerActivity。

代码清单11-5　添加CrimePagerActivity到manifest配置文件（AndroidManifest.xml）

```
<manifest ...>
    ...
    <application ...>
    ...
        <activity
            android:name=".CrimeActivity"
            android:name=".CrimePagerActivity">
        </activity>
```

最后，从项目工具窗口中删除CrimeActivity.java文件，让项目代码更干净。

运行CriminalIntent应用。点击Crime #0查看其明细，然后左右滑屏浏览其他crime明细。可以看到，页面切换流畅，数据加载毫无延迟。ViewPager默认加载当前屏幕上的列表项，以及左右相邻页面的数据，因此响应迅速。如果有需要，可以调用setOffscreenPageLimit(int)方法，定制预加载相邻页面的数目。

注意，目前ViewPager还不够完美。单击后退键返回列表项界面，再点选其他crime列表项。可以看到屏幕上显示的仍是第一个crime列表项的明细，而非当前点选的列表项。

ViewPager默认只显示PagerAdapter中的第一个列表项。要显示选中的列表项，可设置

ViewPager当前要显示的列表项为crime数组中指定位置的列表项。

在CrimePagerActivity.onCreate(Bundle)方法的末尾，循环检查crime的ID，找到所选crime在数组中的索引位置。如果Crime实例的mId与intent extra的crimeId相匹配，设置显示指定位置的列表项，如代码清单11-6所示。

代码清单11-6　设置初始分页显示项（CrimePagerActivity.java）

```
public class CrimePagerActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {
            ...
        });

        for (int i = 0; i < mCrimes.size(); i++) {
            if (mCrimes.get(i).getId().equals(crimeId)) {
                mViewPager.setCurrentItem(i);
                break;
            }
        }
    }
}
```

运行CriminalIntent应用。选择任意列表项，其对应的Crime明细应该能够显示了。现在，ViewPager的使用配置已完成，可以投入使用了。

11.2　FragmentStatePagerAdapter 与 FragmentPagerAdapter

FragmentPagerAdapter是另外一种可用的PagerAdapter，其用法与FragmentStatePagerAdapter基本一致。唯一的区别在于，卸载不再需要的fragment时，各自采用的处理方法有所不同。

FragmentStatePagerAdapter会销毁不需要的fragment。事务提交后，activity的FragmentManager中的fragment会被彻底移除。FragmentStatePagerAdapter类名中的“state”表明：在销毁fragment时，可在onSaveInstanceState(Bundle)方法中保存fragment的Bundle信息。用户切换回来时，保存的实例状态可用来生成新的fragment（如图11-4所示）。

相比之下，FragmentPagerAdapter有不同的做法。对于不再需要的fragment，FragmentPagerAdapter会选择调用事务的detach(Fragment)方法来处理它，而非remove(Fragment)方法。也就是说，FragmentPagerAdapter只是销毁了而fragment的视图，而fragment实例还保留在FragmentManager中。因此，FragmentPagerAdapter创建的fragment永远不会被销毁（如图11-5所示）。

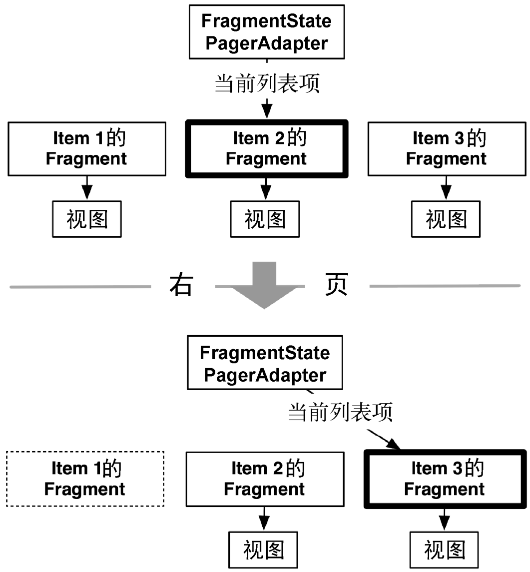


图11-4 `FragmentStatePagerAdapter`的fragment管理

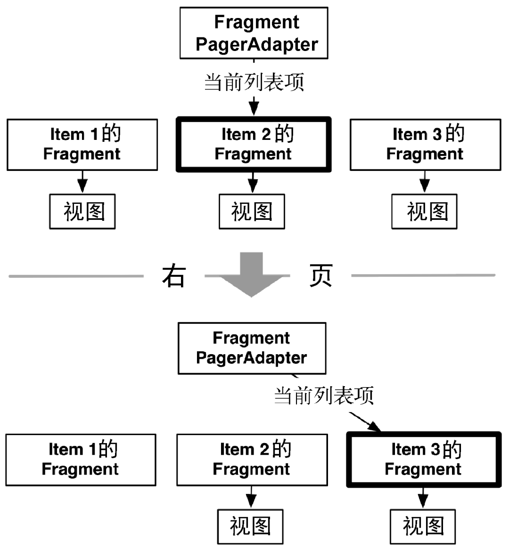


图11-5 `FragmentPagerAdapter`的fragment管理

选择哪种adapter取决于应用的要求。通常来说，使用`FragmentStatePagerAdapter`更节省内存。CriminalIntent应用需显示大量crime记录，每份记录最终还会包含图片。在内存中保存所有信息显然不合适，因此我们选择使用`FragmentStatePagerAdapter`。

另一方面，如果用户界面只需要少量固定的fragment，则FragmentPagerAdapter是安全、合适的选择。最常见的例子为使用tab选项页显示用户界面。例如，某些应用的明细视图所含内容较多，通常需分两页显示。这时就可以将这些明细信息分拆开来，以多页面的形式展现。显然，为用户界面添加支持滑动切换的ViewPager，能增强应用的触摸体验。此外，将fragment保存在内存中，更易于管理控制器层的代码。对于这种类型的用户界面，每个activity通常只有两三个fragment，基本不用担心有内存不足的风险。

11.3　深入学习：ViewPager 的工作原理

ViewPager和PagerAdapter在后台为我们完成了很多工作。本节我们来深入学习ViewPager的工作原理。

继续之前，先提个醒：大多数情况下，我们无需了解其内部实现细节。

不过，如果要自己实现PagerAdapter接口，则需了解ViewPager-PagerAdapter和Recycler-View-Adapter各自关系的异同。

什么时候需要自己实现PagerAdapter接口呢？需要ViewPager托管非fragment视图时（如图片这样的常见View对象），就需要实现原生PagerAdapter接口。

为什么选择使用ViewPager而不是RecyclerView呢？

由于无法使用现有的Fragment，因此在CriminalIntent应用中使用RecyclerView需处理大量内部实现工作。Adapter需要我们及时地提供View。然而，决定fragment视图何时创建的是FragmentManager。因此，当RecyclerView要求Adapter提供fragment视图时，我们无法立即创建fragment并提供其视图。

这就是ViewPager存在的理由。它使用的是PagerAdapter类，而非原来的Adapter。PagerAdapter要比Adapter复杂得多，因为它要处理更多的视图管理工作。以下为它的内部实现。

PagerAdapter不使用可返回视图的onBindViewHolder(...)方法，而是使用下列方法：

```
public Object instantiateItem(ViewGroup container, int position)
public void destroyItem(ViewGroup container, int position, Object object)
public abstract boolean isViewFromObject(View view, Object object)
```

PagerAdapter.instantiateItem(ViewGroup, int)方法告诉pager adapter创建指定位置的列表项视图，然后将其添加给ViewGroup视图容器，而destroyItem(ViewGroup, int, Object)方法则告诉pager adapter销毁已建视图。注意，instantiateItem(ViewGroup, int)方法并不要求立即创建视图。因此，PagerAdapter可自行决定何时创建视图。

视图创建完成后，ViewPager会在某个时间点看到它。为确定该视图所属的对象，ViewPager会调用isViewFromObject(View, Object)方法。这里，Object参数是instantiateItem(ViewGroup,int)方法返回的对象。因此，假设ViewPager调用instantiateItem(ViewGroup, 5)方法返回A对象，那么只要传入的View参数是第5个对象的视图，isViewFromObject(View, A)方法就应返回true值，否则返回false值。

对于ViewPager来说，这是一个复杂的过程，但对于PagerAdapter来说，这算不上什么，因为PagerAdapter只要能够创建、销毁视图以及识别视图来自哪个对象即可。这样的要求显然很宽松，因而PagerAdapter能够比较自由地通过instantiateItem(ViewGroup, int)方法创建并添加新的fragment，然后返回可以跟踪管理的Object（fragment）。以下为isViewFromObject(View, Object)方法的具体实现：

```
@Override
public boolean isViewFromObject(View view, Object object) {
    return ((Fragment)object).getView() == view;
}
```

可以看到，每次需要使用ViewPager时，都要覆盖实现PagerAdapter的这些方法，这真是一种磨难。幸好，还有FragmentPagerAdapter和FragmentStatePagerAdapter。真心感谢它们！

11.4 深入学习：以代码的方式创建视图

本书一直通过布局文件创建视图，其实也可以在代码里创建视图。

实际上，完全可以不用创建布局文件，而使用以下代码定义ViewPager。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ViewPager viewPager = new ViewPager(this);
    setContentView(viewPager);
    ...
}
```

以代码的方式创建视图很简单：调用视图类的构造方法，并传入Context参数。不创建任何布局文件，用代码就能创建完整的视图层级结构。

11

但最好不要这样做，下面就来谈谈使用布局文件的好处。

布局文件能很好地分离控制器层和视图层对象：视图定义在XML布局文件中，控制器层对象定义在Java代码中。这样，假设控制器层有代码修改的话，代码变更管理相对容易很多；反之亦然。

另外，使用布局文件，我们还能使用Android的资源适配系统，实现按设备属性自动调用合适的布局文件（如第3章中横屏模式的布局文件）。

当然，布局文件也不是毫无缺点。如果应用只需一个视图，估计没人愿意麻烦地创建并实例化布局XML文件。

除此之外，创建布局文件的缺点都不值得一提。要知道，Android开发团队从没提倡过以代码的方式创建视图，即便是在因设备配置低、开发人员绞尽脑汁地提高应用性能的年代。因此，使用布局文件吧，哪怕只是添加一个小小的视图ID也会更加简单！

11.5 挑战练习：恢复 CrimeFragment 的边距

可能你已经注意到了，CrimeFragment的边距没有了。奇怪啊，在fragment_crime.xml文件

里，明明已指定过16dp的边距：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="16dp"
    android:orientation="vertical">
```

发生了什么？原来，ViewPager的布局参数是不支持边距设置的。请修改fragment_crime.xml布局文件，让边距能够显示出来。

11.6　挑战练习：添加 Jump to First 按钮和 Jump to Last 按钮

给CrimePagerActivity添加两个按钮。允许使用它们快速跳至第一条和最后一条crime记录。当然，要注意控制，查看第一条记录时应禁用Jump to First按钮，查看最后一条时禁用Jump to Last按钮。

第 12 章　对　话　框

对话框既能引起用户的注意也可接收用户的输入。在提示重要信息或提供用户选项方面，它都非常有用。本章，我们为Criminal Intent应用添加对话框，以便用户修改crime记录日期。用户点击`CrimeFragment`中的日期按钮时，应用会弹出对话框（Lollipop及以上版本），如图12-1所示。

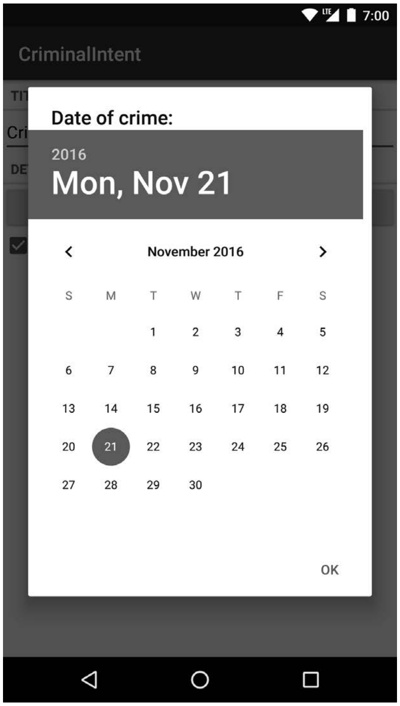


图12-1　可选择crime日期的对话框

图12-1中的对话框是`AlertDialog`类的一个实例。`AlertDialog`类是常用的多用途`Dialog`子类。

Google针对Lollipop系统重新设计了对话框。相比旧系统版本的对话框（参见图12-2左侧），新版对话框界面看起来漂亮多了。

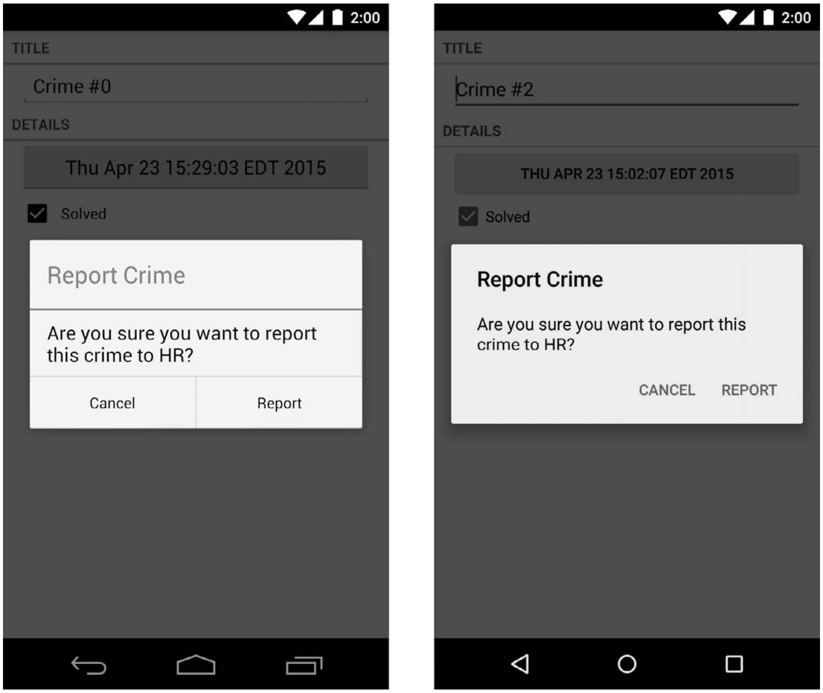


图12-2　旧系统（左）与新系统的对话框界面对比

如果所有Android用户都能用上新式对话框该多好！Google也是这么想的，因而提供了AppCompat库版AlertDialog类。这个类和操作系统内置版AlertDialog类似，能兼容旧版本系统。要使用它，你需要引入android.support.v7.app.AlertDialog依赖项。

12.1　创建 DialogFragment

建议将AlertDialog封装在DialogFragment（Fragment的子类）实例中使用。当然，不使用DialogFragment也可显示AlertDialog视图，但不推荐这样做。使用FragmentManager管理对话框，可以更灵活地显示对话框。

另外，如果旋转设备，单独使用的AlertDialog会消失，而封装在fragment中的AlertDialog则不会有此问题（旋转后，对话框会被重建恢复）。

就CriminalIntent应用来说，我们首先会创建名为DatePickerFragment的DialogFragment子类。然后，在DatePickerFragment中，创建并配置显示DatePicker组件的AlertDialog实例。DatePickerFragment同样由CrimePagerActivity托管。

图12-3展示了以上各对象间的关系。

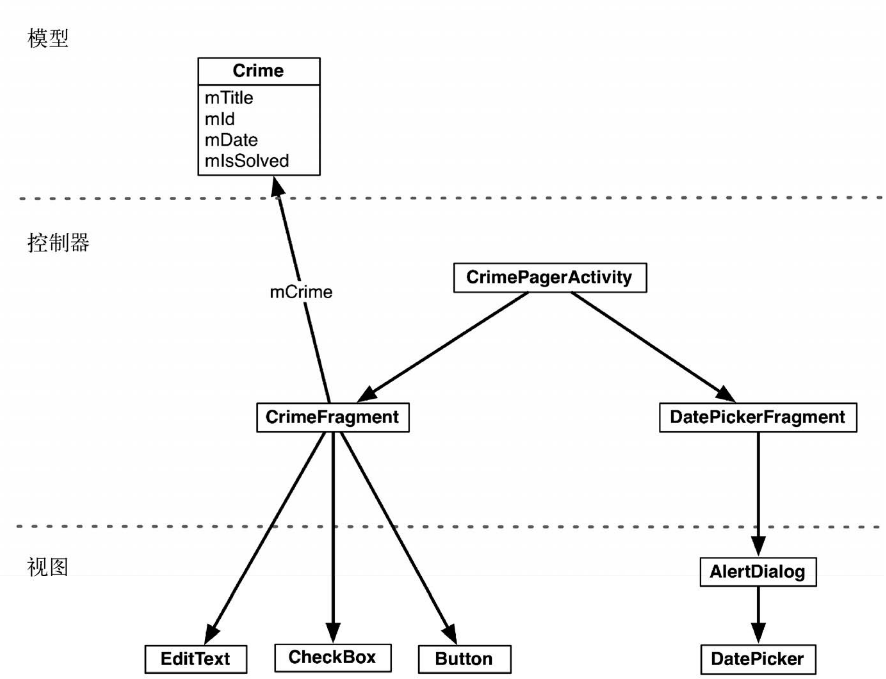


图12-3 由CrimePagerActivity托管的两个fragment对象

要显示对话框，首先应完成以下任务：

- ❑ 创建DatePickerFragment类；
- ❑ 创建AlertDialog；
- ❑ 借助FragmentManager在屏幕上显示对话框。

稍后，我们还将配置使用DatePicker，并实现在CrimeFragment和DatePickerFragment之间传递数据。

继续学习之前，请参照代码清单12-1添加字符串资源备用。

代码清单12-1 为对话框标题添加字符串资源（values/strings.xml）

```
<resources>
    ...
    <string name="crime_solved_label">Solved</string>
    <string name="date_picker_title">Date of crime:</string>

</resources>
```

创建DatePickerFragment新类，并设置其DialogFragment超类为支持库中的android.support.v4.app.DialogFragment类。

DialogFragment类有如下方法：

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

为了在屏幕上显示DialogFragment，托管activity的FragmentManager会调用它。

在DatePickerFragment.java中，添加onCreateDialog(Bundle)方法的实现代码，创建一个带标题栏和OK按钮的AlertDialog，如代码清单12-2所示。（DatePicker组件稍后添加。）

导入AlertDialog时，请确认选择的是AppCompat库中的版本：android.support.v7.app.AlertDialog。

代码清单12-2　创建DialogFragment（DatePickerFragment.java）

```
public class DatePickerFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
    }
}
```

以上代码中，我们使用AlertDialog.Builder类，以流接口的方式创建了AlertDialog实例。下面详细解读一下这段代码。

首先，将Context参数传入AlertDialog.Builder类的构造方法，返回一个AlertDialog.Builder实例。

然后，调用以下两个AlertDialog.Builder方法，配置对话框：

```
public AlertDialog.Builder setTitle(int titleId)
public AlertDialog.Builder setPositiveButton(int textId,
    DialogInterface.OnClickListener listener)
```

调用setPositiveButton(...)方法，需传入两个参数：字符串资源和实现DialogInterface.OnClickListener接口的对象。代码清单12-2中传入的资源ID是Android的OK常量；至于监听器参数，暂时传入null值。监听器接口稍后实现。

（Android有3种可用于对话框的按钮：positive按钮、negative按钮以及neutral按钮。用户点击positive按钮接受对话框展现信息。如果同一对话框上放置有多个按钮，按钮的类型与命名决定着它们在对话框上显示的位置。）

最后，调用AlertDialog.Builder.create()方法，返回配置完成的AlertDialog实例，完成对话框的创建。

使用AlertDialog和AlertDialog.Builder，还可实现更多个性化的需求，请查阅开发者文档详细了解。接下来，我们学习如何在屏幕上显示对话框。

12.1.1　显示 DialogFragment

和其他fragment一样，DialogFragment实例也是由托管activity的FragmentManager管理的。

要将DialogFragment添加给FragmentManager管理并放置到屏幕上，可调用fragment实例

的以下方法：

```
public void show(FragmentManager manager, String tag)
public void show(FragmentTransaction transaction, String tag)
```

String参数可唯一识别FragmentManager队列中的DialogFragment。两个方法都可以：如果传入FragmentTransaction参数，你自己负责创建并提交事务；如果传入FragmentManager参数，系统会自动创建并提交事务。

这里，我们选择传入FragmentManager参数。

在CrimeFragment中，为DatePickerFragment添加一个tag常量。然后，在onCreateView(...)方法中，删除禁用日期按钮的代码。为mDateButton按钮添加OnClickListener监听器接口，实现点击日期按钮展现DatePickerFragment界面，如代码清单12-3所示。

代码清单12-3 显示DialogFragment（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        mDateButton = (Button) v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);
        mDateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = new DatePickerFragment();
                dialog.show(manager, DIALOG_DATE);
            }
        });

        mSolvedCheckBox = (CheckBox) v.findViewById(R.id.crime_solved);
        ...
        return v;
    }
}
```

运行CriminalIntent应用。点击日期按钮弹出对话框，如图12-4所示。

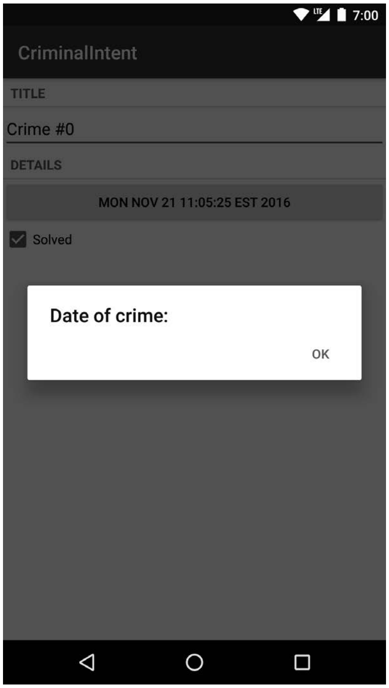


图12-4　带标题和OK按钮的`AlertDialog`

12.1.2　设置对话框的显示内容

接下来，使用`AlertDialog.Builder`的`setView(...)`方法，给`AlertDialog`对话框添加`DatePicker`组件：

```
public AlertDialog.Builder setView(View view)
```

该方法配置对话框，实现在标题栏与按钮之间显示传入的`View`对象。

在项目工具窗口中，以`DatePicker`为根元素，创建名为dialog_date.xml的布局文件。新布局仅包含一个`View`对象，即我们生成并传给`setView(...)`方法的`DatePicker`视图。

参照图12-5，配置`DatePicker`布局。

```
DatePicker
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/dialog_date_picker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:calendarViewShown="false"
```

图12-5　`DatePicker`布局（layout/dialog_date.xml）

在DatePickerFragment.onCreateDialog(Bundle)方法中，实例化DatePicker视图并添加给对话框，如代码清单12-4所示。

代码清单12-4　给AlertDialog添加DatePicker（DatePickerFragment.java）

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = LayoutInflater.from(getActivity())
        .inflate(R.layout.dialog_date, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}
```

运行CriminalIntent应用。点击日期按钮，确认能在对话框中显示DatePicker视图。如果是Lollipop系统，还能看到日历选择界面，如图12-6所示。

图12-6所示的日历选择界面是随material design引入的。这个版本的DatePicker组件会忽略布局中指定的calendarViewShown属性。如果使用旧版本系统，DatePicker组件会使用calendarViewShown属性，因而我们会看到如图12-7所示的用户界面。

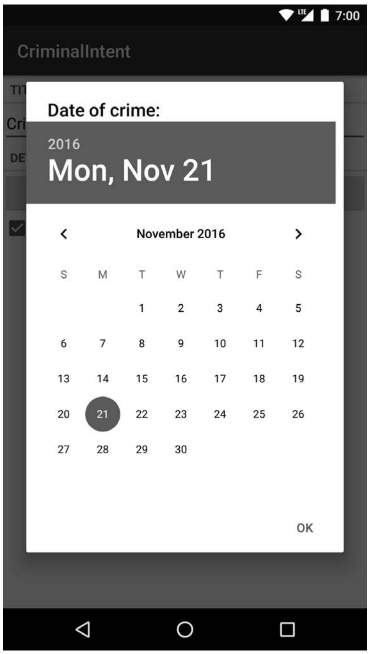


图12-6　Lollipop系统中的DatePicker



图12-7　显示DatePicker的AlertDialog

对话框完全兼容新旧系统，不过新系统版本的界面显然更美观。

采用以下代码也能创建DatePicker对象，为何还要费事地定义XML布局文件，再去实例化视图对象呢？

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    DatePicker datePicker = new DatePicker(getActivity());

    return new AlertDialog.Builder(getActivity())
        .setView(datePicker)
        ...
        .create();
}
```

这是因为，想调整对话框的显示内容时，直接修改布局文件会更容易些。例如，如果想在对话框的DatePicker旁再添加一个TimePicker，只需更新布局文件就能显示新视图。

即使设备旋转，用户所选日期也都会得到保留（试试看）。这是如何做到的呢？前面说过，设备配置改变时，具有ID属性的视图可以保存运行状态；而我们以dialog_date.xml布局创建DatePicker时，编译工具已为DatePicker生成了唯一的ID。

如果以代码的方式创建DatePicker，想看到同样的效果，需要为其设置ID属性。

至此，显示对话框的工作就完成了。下一节，我们实现显示Crime日期，并支持用户对其进行修改。

12.2　fragment 间的数据传递

前面，我们实现了activity之间以及基于fragment的activity之间的数据传递。现在需实现同一activity托管的两个fragment之间的数据传递（CrimeFragment和DatePickerFragment），如图12-8所示。

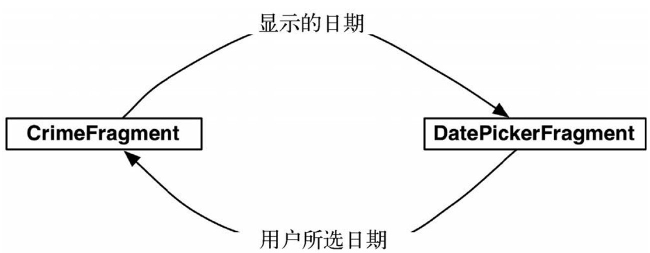


图12-8　CrimeFragment与DatePickerFragment间的对话

要传递crime的日期给DatePickerFragment，需新建一个newInstance(Date)方法，然后将Date作为argument附加给fragment。

为返回新日期给CrimeFragment，并更新模型层以及对应视图，需将日期打包为extra并附加到Intent上，然后调用CrimeFragment.onActivityResult(...)方法，并传入准备好的Intent参数，如图12-9所示。

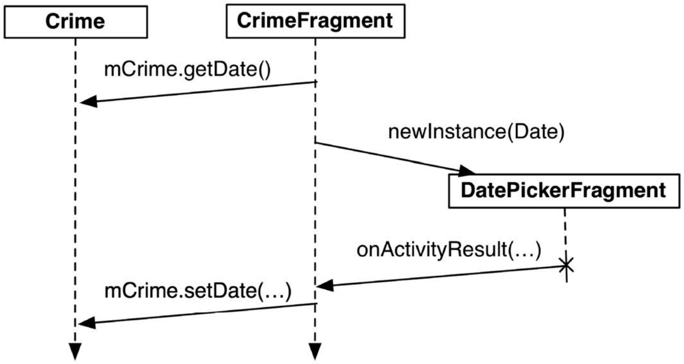


图12-9　CrimeFragment和DatePickerFragment间的事件流

在稍后的实现代码中可以看到，我们没有调用托管activity的Activity.onActivityResult(...)方法，而是调用了Fragment.onActivityResult(...)方法，这似乎令人费解。实际上，调用onActivityResult(...)方法实现fragment间的数据传递不仅行得通，而且可以更灵活地展现对话框fragment（稍后会看到）。

12.2.1　传递数据给 DatePickerFragment

要传递crime日期给DatePickerFragment，需将它保存在DatePickerFragment的argument bundle中。这样，DatePickerFragment就能直接获取它。

创建和设置fragment argument通常是在newInstance()方法中完成的（代替fragment构造方法）。在DatePickerFragment.java中，添加一个newInstance(Date)方法，如代码清单12-5所示。

代码清单12-5　添加newInstance(Date)方法（DatePickerFragment.java）

```
public class DatePickerFragment extends DialogFragment {

    private static final String ARG_DATE = "date";

    private DatePicker mDatePicker;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_DATE, date);

        DatePickerFragment fragment = new DatePickerFragment();
        fragment.setArguments(args);
        return fragment;
    }
    ...
}
```

然后，在CrimeFragment中，用DatePickerFragment.newInstance(Date)方法替换DatePickerFragment的构造方法，如代码清单12-6所示。

12

代码清单12-6　添加newInstance()方法（CrimeFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater,ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setText(mCrime.getDate().toString());
    mDateButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            FragmentManager manager = getFragmentManager()
            DatePickerFragment dialog = new DatePickerFragment();
            DatePickerFragment dialog = DatePickerFragment
                .newInstance(mCrime.getDate());
            dialog.show(manager, DIALOG_DATE);
        }
    });
    ...
    return v;
}
```

DatePickerFragment使用Date中的信息来初始化DatePicker对象。然而，DatePicker对象的初始化需整数形式的月、日、年。Date是时间戳，无法直接提供整数。

要达到目的，必须首先创建一个Calendar对象，然后用Date对象配置它，再从Calendar对象中取回所需信息。

在onCreateDialog(Bundle)方法内，从argument中获取Date对象，然后用它和Calendar对象初始化DatePicker，如代码清单12-7所示。

代码清单12-7　获取Date对象并初始化DatePicker（DatePickerFragment.java）

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    Date date = (Date) getArguments().getSerializable(ARG_DATE);

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = LayoutInflater.from(getActivity())
            .inflate(R.layout.dialog_date, null);

    mDatePicker = (DatePicker) v.findViewById(R.id.dialog_date_picker);
    mDatePicker.init(year, month, day, null);

    return new AlertDialog.Builder(getActivity())
            .setView(v)
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
}
```

现在，我们成功完成了CrimeFragment向DatePickerFragment传递日期。运行CriminalIntent应用，看看效果如何。

12.2.2 返回数据给 CrimeFragment

要让CrimeFragment接收DatePickerFragment返回的日期数据，首先需要清楚它们之间的关系。

如果是activity的数据回传，我们调用startActivityForResult(...)方法，ActivityManager负责跟踪管理activity父子关系。回传数据后，子activity被销毁，但ActivityManager知道接收数据的是哪个activity。

1. 设置目标fragment

类似于activity间的关联，可将CrimeFragment设置成DatePickerFragment的目标fragment。这样，在CrimeFragment和DatePickerFragment被销毁并重建后，操作系统会重新关联它们。调用以下Fragment方法可建立这种关联：

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

该方法有两个参数：目标fragment以及类似于传入startActivityForResult(...)方法的请求代码。需要时，目标fragment使用请求代码确认是哪个fragment在回传数据。

目标fragment和请求代码由FragmentManager负责跟踪管理，我们可调用fragment（设置目标fragment的fragment）的getTargetFragment()方法和getTargetRequestCode()方法获取它们。

在CrimeFragment.java中，创建请求代码常量，然后将CrimeFragment设为DatePickerFragment实例的目标fragment，如代码清单12-8所示。

12

代码清单12-8 设置目标fragment（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";

    private static final int REQUEST_DATE = 0;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        mDateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = DatePickerFragment
                        .newInstance(mCrime.getDate());
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
                dialog.show(manager, DIALOG_DATE);
```

```
            }
        });
        ...
        return v;
    }
}
```

2. 传递数据给目标fragment

建立CrimeFragment与DatePickerFragment之间的联系后，需将数据回传给CrimeFragment。回传日期将作为extra附加给Intent。

使用什么方法发送intent信息给目标fragment？虽然令人难以置信，但是我们会让DatePickerFragment类调用CrimeFragment.onActivityResult(int, int, Intent)方法。

Activity.onActivityResult(...)方法是ActivityManager在子activity被销毁后调用的父activity方法。处理activity间的数据返回时，ActivityManager会自动调用Activity.onActivityResult(...)方法。父activity接收到Activity.onActivityResult(...)方法调用命令后，其FragmentManager会调用对应fragment的Fragment.onActivityResult(...)方法。

处理由同一activity托管的两个fragment间的数据返回时，可借用Fragment.onActivityResult(...)方法。因此，直接调用目标fragment的Fragment.onActivityResult(...)方法，就能实现数据的回传。该方法恰好有我们需要的如下信息。

- 请求代码：与传入setTargetFragment(...)方法的代码相匹配，告诉目标fragment返回结果来自哪里。
- 结果代码：决定下一步该采取什么行动。
- Intent：包含extra数据。

在DatePickerFragment类中，新建sendResult(...)私有方法，创建intent并将日期数据作为extra附加到intent上。最后调用CrimeFragment.onActivityResult(...)方法，如代码清单12-9所示。

代码清单12-9　回调目标fragment（DatePickerFragment.java）

```
public class DatePickerFragment extends DialogFragment {

    public static final String EXTRA_DATE =
            "com.bignerdranch.android.criminalintent.date";

    private static final String ARG_DATE = "date";
    ...
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        ...
    }

    private void sendResult(int resultCode, Date date) {
        if (getTargetFragment() == null) {
            return;
        }
```

```
            Intent intent = new Intent();
            intent.putExtra(EXTRA_DATE, date);

            getTargetFragment()
                    .onActivityResult(getTargetRequestCode(), resultCode, intent);
        }
    }
```

现在来使用sendResult(...)私有方法。用户点击对话框中的positive按钮时，需要从DatePicker中获取日期并回传给CrimeFragment。在onCreateDialog(...)方法中，替换掉setPositiveButton(...)的null参数值，实现DialogInterface.OnClickListener监听器接口。在监听器接口的onClick(...)方法中，获取日期并调用sendResult(...)方法，如代码清单12-10所示。

代码清单12-10 一切是否都OK？（DatePickerFragment.java）

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    ...
    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null);
        .setPositiveButton(android.R.string.ok,
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        int year = mDatePicker.getYear();
                        int month = mDatePicker.getMonth();
                        int day = mDatePicker.getDayOfMonth();
                        Date date = new GregorianCalendar(year, month, day).getTime();
                        sendResult(Activity.RESULT_OK, date);
                    }
        })
        .create();
}
```

12

在CrimeFragment中，覆盖onActivityResult(...)方法，从extra中获取日期数据，设置对应Crime的记录日期，然后刷新日期按钮的显示，如代码清单12-11所示。

代码清单12-11 响应DatePicker对话框（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
```

```
                return;
            }

            if (requestCode == REQUEST_DATE) {
                Date date = (Date) data
                        .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
                mCrime.setDate(date);
                mDateButton.setText(mCrime.getDate().toString());
            }
        }
    }
```

在onCreateView(...)和onActivityResult(...)这两个方法中，设置按钮显示文字的代码完全一样。为了避免代码冗余，可以将其封装到updateDate()私有方法中，然后分别调用。

除手动封装代码的方式外，还可以使用Android Studio的内置工具。高亮选取设置mDateButton显示文字的代码，如代码清单12-12所示。

代码清单12-12　高亮选取日期按钮更新代码（CrimeFragment.java）

```
@Override
public void onActivityResult(int requestCode, int resultCode, intent data) {
    if (resultCode != Activity.RESUlT_OK) {
        return;
    }

    if (requestCode == REQUEST_DATE) {
        Date date = (Date) data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        mDateButton.setText(mCrime.getDate().toString());
    }
}
```

右键单击并选择Refactor → Extract → Method...菜单项，弹出如图12-10所示的界面。

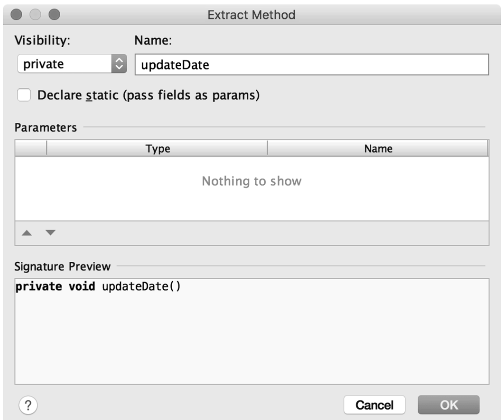


图12-10　使用Android Studio抽取方法

设置方法为私有并将其命名为updateDate。点击OK按钮，Android Studio会提示还有其他地方使用了这段代码。点击Yes允许它自动处理。然后确认updateDate方法封装完成并在相应地方已调用，如代码清单12-13所示。

代码清单12-13 使用updateDate()私有方法（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        ...
        mDateButton = (Button) v.findViewById(R.id.crime_date);
        updateDate();
        ...
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }

        if (requestCode == REQUEST_DATE) {
            Date date = (Date) data
                    .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            updateDate();
        }
    }

    private void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
    }
}
```

日期数据的双向传递完成了。运行CriminalIntent应用，确保可以控制日期的传递与显示。修改某项Crime的日期，确认CrimeFragment视图显示了新日期。然后返回crime列表项界面，查看对应Crime的日期，并确认模型层数据已得到更新。

3. 更为灵活的DialogFragment视图展现

编写需要用户大量输入以及要求更多空间显示输入的应用，并且要让应用同时支持手机和平板设备时，使用onActivityResult(...)方法返回数据给目标fragment是比较方便的。

手机屏幕空间有限，因此通常需要使用activity托管全屏的fragment界面，以显示用户输入要求。该子activity会由父activity的fragment以调用startActivityForResult(...)方法的方式启动。子activity被销毁后，父activity会接收到onActivityResult(...)方法的调用请求，并将之转发给启动子activity的fragment，如图12-11所示。

12

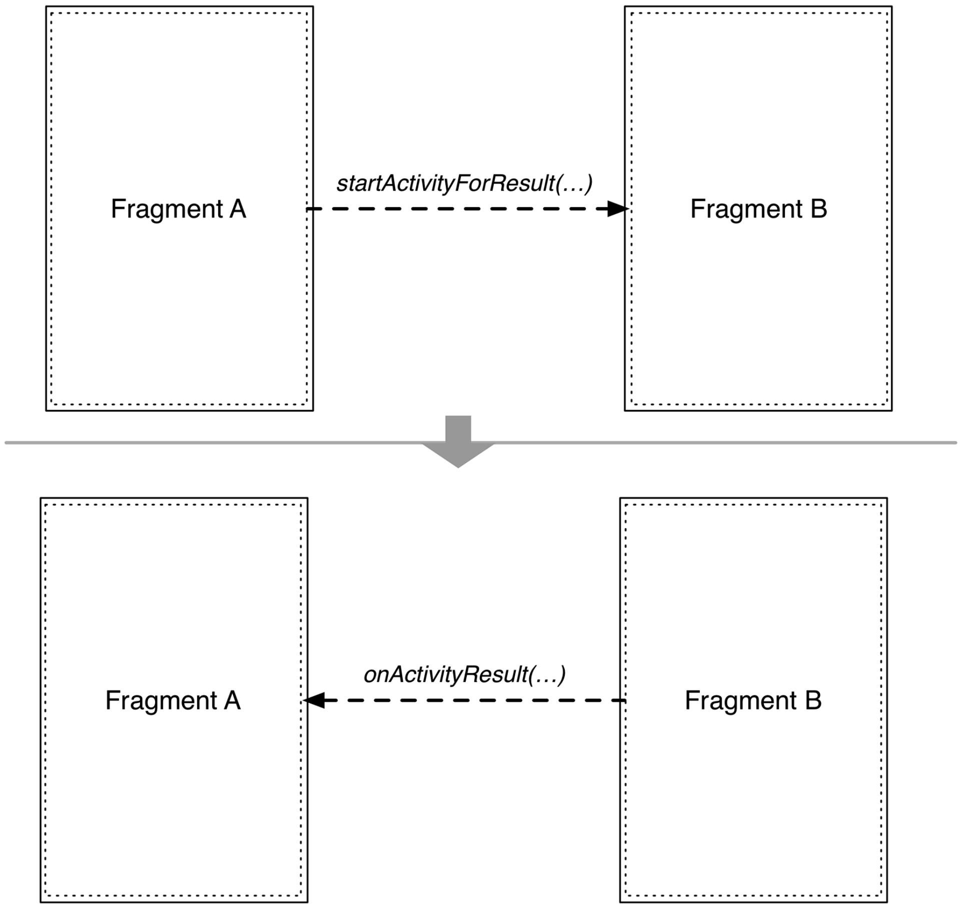


图12-11　手机设备上activity间的数据传递

平板设备的屏幕比较大，适合以弹出对话框的方式显示信息和接收用户输入。这种情况下，应设置目标fragment并调用对话框fragment的`show(...)`方法。对话框被取消后，对话框fragment会调用目标fragment的`onActivityResult(...)`方法，如图12-12所示。

无论是启动子activity还是显示对话框，fragment的`onActivityResult(...)`方法总会被调用。因此，可使用相同的代码实现不同的信息呈现。

编写同样的代码用于全屏fragment或对话框fragment时，可选择覆盖`DialogFragment.onCreateView(...)`方法，而非`onCreateDialog(...)`方法，以实现不同设备上的信息呈现。

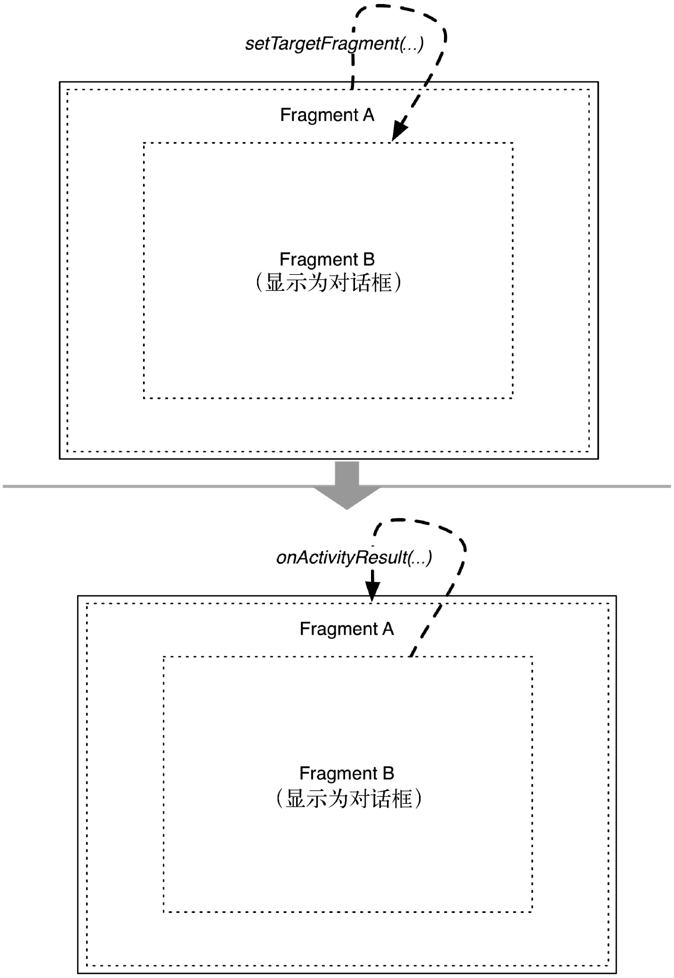


图12-12 平板设备上fragment间的数据传递

12.3 挑战练习：更多对话框

首先看一个简单的练习。另写一个名为TimePickerFragment的对话框fragment，允许用户使用TimePicker组件选择crime发生的具体时间。在CrimeFragment用户界面上再添加一个按钮，以显示TimePickerFragment视图界面。

12.4 挑战练习：实现响应式 DialogFragment

再来看一个有些难度的练习：优化DatePickerFragment的呈现方式。

要完成这个挑战，初步分析需三大步。第一步，替换掉onCreateDialog(Bundle)方法，改用onCreateView(...)方法来创建DatePickerFragment的视图。以这种方式创建DialogFragment的话，在对话框界面上看不到标题区域，同样也没有放置按钮的空间。这需要你自行在dialog_date.xml布局中创建OK按钮。

有了DatePickerFragment视图，接下来就能以对话框或以在activity中内嵌的方式展现。第二步，创建SingleFragmentActivity子类。它的任务就是托管DatePickerFragment。

选择这种方式展现DatePickerFragment，就要使用startActivityForResult(...)方法回传日期给CrimeFragment。在DatePickerFragment中，如果目标fragment不存在，就调用托管activity的setResult(int, intent)方法回传日期给CrimeFragment。

最后，修改CriminalIntent应用：如果是手机设备，就以全屏activity的方式展现DatePickerFragment；如果是平板设备，就以对话框的方式展现DatePickerFragment。想知道如何按设备屏幕大小优化应用，请提前学习第17章的相关内容。

第 13 章 工　具　栏

优秀的Android应用都注重工具栏设计。工具栏可放置菜单选项、提供应用导航，还能帮助统一设计风格、塑造品牌形象。

本章，我们为CriminalIntent应用创建工具栏菜单，提供新增crime记录的菜单项，并实现向上按钮的导航功能，如图13-1所示。

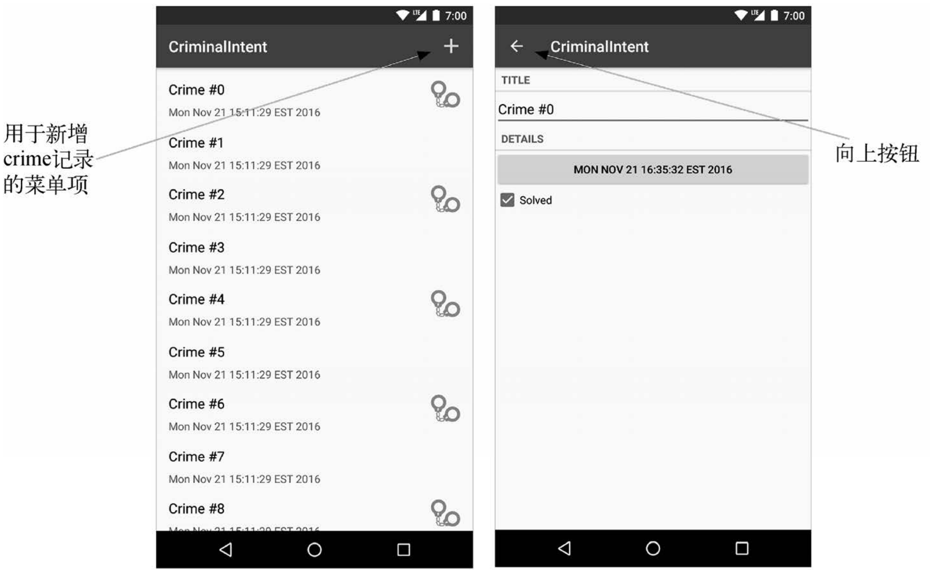


图13-1　CriminalIntent应用的工具栏

13.1　AppCompat

Android 5.0（Lollipop）引入了工具栏这个新增组件。在Lollipop之前，应用中用于导航或提供菜单操作的是操作栏。

工具栏和操作栏有些类似，但它基于操作栏进化而来。因此，工具栏的界面更美观，使用更方便。

CriminalIntent应用最低只支持API 19级，原生工具栏无法支持更老版本的系统。不过，Google已将它移植到了AppCompat库。这样一来，老版本系统（API 9级、Android 2.3以上）就都能使用Lollipop上的工具栏了。

使用 AppCompat 库

我们已经用过AppCompat库。本书撰写时，新项目都会默认使用这个库。如果想给老项目添加AppCompat库，该如何做呢？这需要你做几件事情。

要整合使用AppCompat库，你需要：

- ❑ 添加AppCompat依赖项；
- ❑ 使用一种AppCompat主题；
- ❑ 确保所有activity都是`AppCompatActivity`子类。

1. 更新主题

在第7章，我们已经为CriminalIntent项目添加过依赖项，接下来至少要使用一种AppCompat主题。AppCompat库自带以下三种主题。

- ❑ `Theme.AppCompat`：黑色主题
- ❑ `Theme.AppCompat.Light`：浅色主题
- ❑ `Theme.AppCompat.Light.DarkActionBar`：带黑色工具栏的浅色主题

应用级别的主题设置在AndroidManifest.xml文件中进行。主题也可按activity配置。打开AndroidManifest.xml文件，查看`application`标签的`android:theme`属性，应该可以看到如代码清单13-1所示的主题配置。

代码清单13-1　各种manifest配置项（AndroidManifest.xml）

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme" >
```

`AppTheme`定义在res/values/styles.xml文件中。打开这个文件，参照代码清单13-2设置应用的主题。这个文件里还有一些其他属性，暂时不用理会，后面会更新。

代码清单13-2　使用AppCompat主题（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.light.DarkActionBar">
      <!-- Customize your theme here. -->
      <item name="colorPrimary">@color/colorPrimary</item>
      <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
      <item name="colorAccent">@color/colorAccent</item>
    </style>
```

```
</resources>
```

在第22章，我们还会进一步学习与样式和主题相关的知识。

2. 使用AppCompatActivity

最后一步是让activity类继承AppCompatActivity类。自第7章开始，我们就已经在用AppCompatActivity，所以这步略过。

运行CriminalIntent应用。若一切正常，应该可以看到如图13-2所示的画面。



图13-2　换了主题的工具栏

现在，可以着手添加工具栏菜单了。

13.2　工具栏菜单

工具栏菜单由菜单项（又称操作项）组成，它占据着工具栏的右上方区域。菜单项的操作应用于当前屏幕，甚至整个应用。现在，我们来添加允许用户新增crime记录的菜单项。

菜单及菜单项需用到一些字符串资源。参照代码清单13-3，将它们添加到strings.xml文件中。有些字符串资源现在还用不到，但方便起见，一并完成添加。

代码清单13-3　添加字符串资源（res/values/strings.xml）

```
<resources>
    ...
    <string name="date_picker_title">Date of crime:</string>
    <string name="new_crime">New Crime</string>
    <string name="show_subtitle">Show Subtitle</string>
    <string name="hide_subtitle">Hide Subtitle</string>
```

```
    <string name="subtitle_format">%1$d crimes</string>

</resources>
```

13.2.1 在 XML 文件中定义菜单

菜单是一种类似于布局的资源。创建菜单定义文件并将其放置在res/menu目录下，Android会自动生成相应的资源ID。随后，在代码中实例化菜单时，就可以直接使用。

在项目工具窗口中，右键单击res目录，选择New → Android resource file菜单项。在弹出的窗口界面，选择Menu资源类型，并命名资源文件为fragment_crime_list，点击OK按钮确认，如图13-3所示。

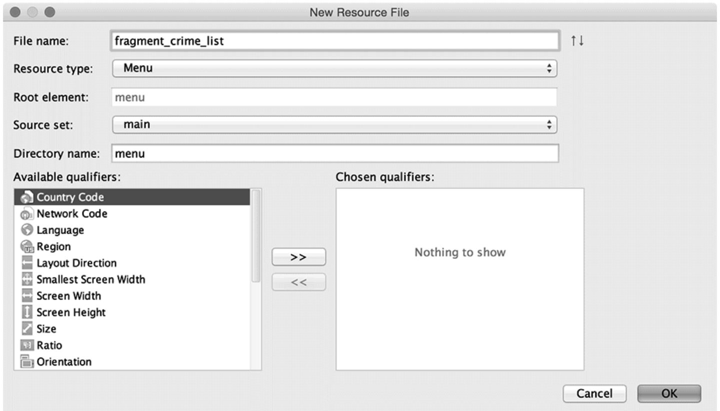


图13-3 创建菜单定义文件

这里，菜单定义文件遵循了与布局文件一样的命名原则。Android Studio会创建res/menu/fragment_crime_list.xml文件。这个文件和`CrimeListFragment`的布局文件同名，但分别位于不同的目录。打开新建的fragment_crime_list.xml文件。参照代码清单13-4，添加新的`item`元素。

代码清单13-4 创建菜单资源（res/menu/fragment_crime_list.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>
</menu>
```

`showAsAction`属性用于指定菜单项是显示在工具栏上，还是隐藏于溢出菜单（overflow menu）。该属性当前设置为`ifRoom`和`withText`的组合值。因此，只要空间足够，菜单项图标及其文字描述都会显示在工具栏上。如果空间仅够显示菜单项图标，文字描述就不会显示。如果空

间大小不够显示任何项，菜单项就会隐藏到溢出菜单中。

如果溢出菜单包含其他项，它们就会以三个点表示（位于工具栏最右端），如图13-4所示。稍后，我们就会更新代码添加这样的菜单项。



图13-4　工具栏中的溢出菜单

属性`showAsAction`还有另外两个可选值：`always`和`never`。不推荐使用`always`，应尽量使用`ifRoom`属性值，让操作系统决定如何显示菜单项。对于那些很少用到的菜单项，`never`属性值是个不错的选择。总之，为了避免用户界面混乱，工具栏上只应放置常用菜单项。

1. app命名空间

注意，不同于常见的`android`命名空间声明，fragment_crime_list.xml文件使用`xmlns`标签定义了全新的`app`命名空间。指定`showAsAction`属性时，就用了这个新定义的命名空间。

出于兼容性考虑，AppCompat库需要使用`app`命名空间。操作栏API随Android 3.0引入。为了支持各种旧系统版本设备，早期创建的AppCompat库捆绑了兼容版操作栏。这样一来，不管新旧，所有设备都能用上操作栏。在运行Android 2.3或更早版本系统的设备上，菜单及其相应的XML文件确实是存在的，但是`android:showAsAction`属性是随着操作栏的发布才添加的。

AppCompat库不希望使用原生`showAsAction`属性，因此，它提供了定制版`showAsAction`属性（`app:showAsAction`）。

2. 使用Android Asset Studio

应用使用的图标有两种：系统图标和项目资源图标。系统图标（system icon）是Android操作系统内置的图标。`android:icon`属性值`@android:drawable/ic_menu_add`就引用了系统图标。

在应用原型设计阶段，使用系统图标不会有什么问题；而在应用发布时，无论用户运行什么设备，最好能统一应用的界面风格。要知道，不同设备或操作系统版本间，系统图标的显示风格差异很大。有些设备的系统图标甚至与应用的整体风格完全不搭。

一种解决方案是创建定制图标。这需要针对不同屏幕显示密度或各种可能的设备配置，准备不同版本的图标。访问developer.android.com/design/style/iconography.html，查看Android的图标设计指南，可了解更多相关信息。

另一种解决方案是找到适合应用的系统图标，将它们直接复制到项目的drawable资源目录中。

系统图标可在Android SDK的安装目录下找到。如果是Mac计算机，路径通常为/Users/user/Library/Android/sdk；如果是Windows计算机，默认的路径是\Users\user\sdk。此外，还可以打开项目结构窗口，选择SDK Location来确认SDK的具体存放路径。

打开SDK目录，可找到包括ic_menu_add在内的Android系统资源。资源的具体目录是/platforms/android-25/data/res，路径中的数字25代表Android的API级别。

还有第三个、也是最容易的解决方案：使用Android Studio内置的Android Asset Studio工具。你可以用它为工具栏创建或定制图片。

在项目工具窗口中，右键单击drawable目录，选择New → Image Asset菜单项，弹出如图13-5所示的Asset Studio窗口。



图13-5　Asset Studio

这里，我们可以生成各类图标。在Icon Type一栏选Action Bar and Tab Icons，在Name一栏输入ic_menu_add，Asset Type处选Clip Art，最后，更新Theme为HOLO_DARK。工具栏使用了深色

系主题，那图标图像就应选浅色。

现在，点击Clip Art按钮挑选剪贴画图片。在弹出的剪贴画窗口，选择看上去像+号的图片，如图13-6所示。

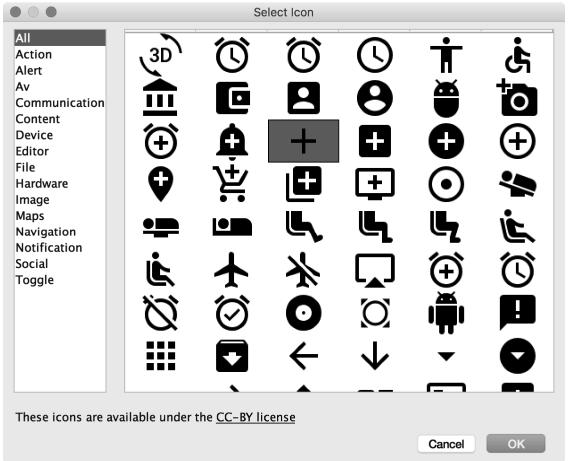


图13-6　可选的剪贴画

点击OK按钮确认，然后单击Next按钮进入如图13-7所示的预览画面。这个预览画面告诉我们，Asset Studio将会产生hdpi、mdpi、xhdpi和xxhdpi类型的图标。真是太方便了！

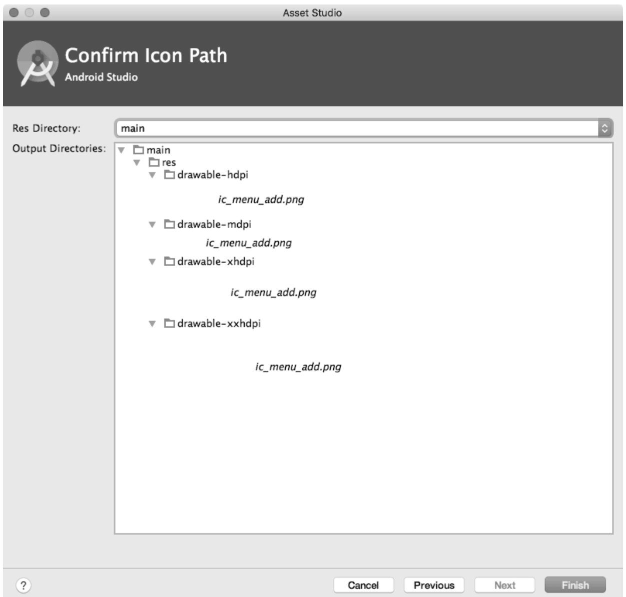


图13-7　Asset Studio生成的文件

最后，点击Finish按钮生成图像。然后，在布局文件中，修改布局文件中的android:icon属性，在项目中使用新图像，如代码清单13-5所示。

代码清单13-5　引用资源（res/menu/fragment_crime_list.xml）

```
<item
    android:id="@+id/new_crime"
    android:icon="@android:drawable/ic_menu_add"
    android:icon="@drawable/ic_menu_add"
    android:title="@string/new_crime"
    app:showAsAction="ifRoom|withText"/>
```

13.2.2　创建菜单

在代码中，Activity类提供了管理菜单的回调函数。需要选项菜单时，Android会调用Activity的onCreateOptionsMenu(Menu)方法。

然而，按照CriminalIntent应用的设计，与选项菜单相关的回调函数需在fragment而非activity里实现。不用担心，Fragment有一套自己的选项菜单回调函数。稍后，我们会在CrimeListFragment中实现这些方法。以下为创建菜单和响应菜单项选择事件的两个回调方法：

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

在CrimeListFragment.java中，覆盖onCreateOptionsMenu(Menu, MenuInflater)方法，实例化fragment_crime_list.xml中定义的菜单，如代码清单13-6所示。

代码清单13-6　实例化选项菜单（CrimeListFragment.java）

```
@Override
public void onResume() {
    super.onResume();
    updateUI();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}
...
```

在以上方法中，我们调用MenuInflater.inflate(int, Menu)方法并传入菜单文件的资源ID，将布局文件中定义的菜单项目填充到Menu实例中。

注意，我们也调用了超类的onCreateOptionsMenu(...)方法。当然，也可以不调，但作为一项开发规范，有理由推荐这么做。调用该超类方法，任何超类定义的选项菜单功能在子类方法中都能获得应用。不过，onCreateOptionsMenu(...)超类方法什么也没做，这里的调用仅仅是遵循约定而已。

Fragment.onCreateOptionsMenu(Menu, MenuInflater)方法是由FragmentManager负责

调用的。因此，当activity接收到操作系统的`onCreateOptionsMenu(...)`方法回调请求时，我们必须明确告诉`FragmentManager`：其管理的fragment应接收`onCreateOptionsMenu(...)`方法的调用指令。要通知`FragmentManager`，需调用以下方法：

```
public void setHasOptionsMenu(boolean hasMenu)
```

定义`CrimeListFragment.onCreate(Bundle)`方法，让`FragmentManager`知道`CrimeList-Fragment`需接收选项菜单方法回调，如代码清单13-7所示。

代码清单13-7　调用setHasOptionsMenu方法（CrimeListFragment.java）

```
public class CrimelistFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
```

运行CriminalIntent应用，查看新创建的菜单项，如图13-8所示。



图13-8　显示在工具栏上的菜单项图标

13

菜单项标题怎么没有显示？大多数手机设备在竖屏模式下屏幕空间有限。因此，应用的工具栏只够显示菜单项图标。长按工具栏上的菜单项图标，可弹出标题，如图13-9所示。



图13-9　长按工具栏上的图标，显示菜单项标题

横屏模式下，工具栏会有足够的空间同时显示菜单项图标和标题，如图13-10所示。

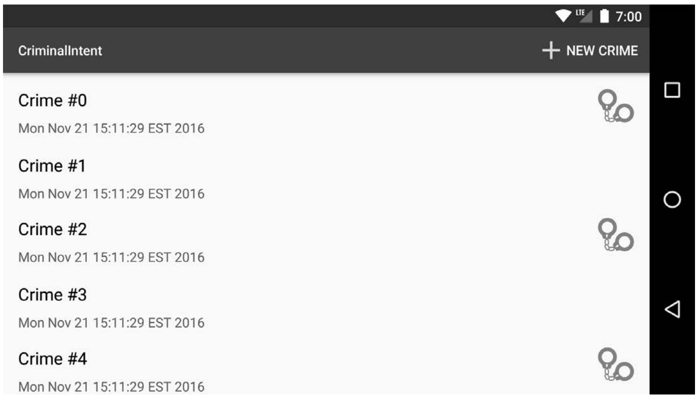


图13-10　同时显示的菜单项图标和标题

13.2.3 响应菜单项选择

为了响应用户点击New Crime菜单项，需实现新方法以向crime列表添加新的Crime。在CrimeLab.java中，新增一个addCrime()方法，如代码清单13-8所示。

代码清单13-8 添加新的crime（CrimeLab.java）

```
...
public void addCrime(Crime c) {
    mCrimes.add(c);
}

public List<Crime> getCrimes() {
    return mCrimes;
}
```

既然可以手动添加crime记录，也就没必要再让程序自动生成100条crime记录了。在CrimeLab.java中，删除生成随机crime记录的代码，如代码清单13-9所示。

代码清单13-9 再见，随机crime记录！（CrimeLab.java）

```
private CrimeLab(Context context) {
    mCrimes = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Crime crime = new Crime();
        crime.setTitle("Crime #" + i);
        crime.setSolved(i % 2 == 0);// Every other one
        mCrimes.add(crime);
    }
}
```

用户点击菜单中的菜单项时，fragment会收到onOptionsItemSelected(MenuItem)方法的回调请求。传入该方法的参数是一个描述用户选择的MenuItem实例。

当前菜单仅有一个菜单项，但菜单通常包含多个菜单项。通过检查菜单项ID，可确定被选中的是哪个菜单项，然后作出相应的响应。这个ID实际就是在菜单定义文件中赋予菜单项的资源ID。

13

在CrimeListFragment.java中，实现onOptionsItemSelected(MenuItem)方法，以响应菜单项的选择事件。在该方法中，创建新的Crime实例，将其添加到CrimeLab中，然后启动CrimePagerActivity实例，让用户可以编辑新创建的Crime记录，如代码清单13-10所示。

代码清单13-10 响应菜单项选择事件（CrimeListFragment.java）

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
```

```
            case R.id.new_crime:
                Crime crime = new Crime();
                CrimeLab.get(getActivity()).addCrime(crime);
                Intent intent = CrimePagerActivity
                        .newIntent(getActivity(), crime.getId());
                startActivity(intent);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

注意，onOptionsItemSelected(MenuItem)方法返回的是布尔值。一旦完成菜单项事件处理，该方法应返回true值以表明任务已完成。另外，默认case表达式中，如果菜单项ID不存在，超类版本方法会被调用。

运行CriminalIntent应用，尝试使用菜单，添加一些crime记录并进行编辑。（新增记录前，空空如也的列表看上去不够专业，不用担心，本章末的挑战练习就是为此而设的。）

13.3　实现层级式导航

目前为止，CriminalIntent应用主要靠后退键在应用内导航。后退键导航又称为*临时性导航*，只能返回到上一次浏览过的用户界面；而*层级式导航*（hierarchical navigation，有时又称为ancestral navigation）可在应用内逐级向上导航。

有了层级式导航，用户可点击工具栏左边的向上按钮向上导航。

打开AndroidManifest.xml，参照代码清单13-11添加parentActivityName属性，开启CriminalIntent应用的层级式导航。

代码清单13-11　启用向上按钮（AndroidManifest.xml）

```
<activity
    android:name=".CrimePagerActivity"
    android:parentActivityName=".CrimeListActivity">
</activity>
```

运行应用并创建新的crime记录。在屏幕的左上方，可看到如图13-11所示的向上按钮。点击按钮可向上一级导航至CrimeListActivity用户界面。

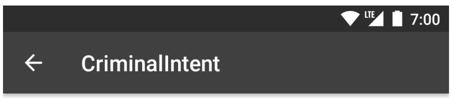


图13-11　CrimePagerActivity界面上的向上按钮

层级式导航的工作原理

CriminalIntent应用中，后退键导航和向上按钮导航执行同样的操作。在CrimePagerActivity

界面，无论按哪个按钮导航，都是回到CrimeListActivity界面。虽然结果一样，但它们各自的后台实现机制大不相同。知道这一点很重要，因为取决于具体应用，向上导航很可能会让用户迷失在众多activity中（这里指回退栈内的众多activity）。

用户点击向上按钮自CrimePagerActivity界面向上导航时，如下的intent会被创建：

```
Intent intent = new Intent(this, CrimeListActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();
```

FLAG_ACTIVITY_CLEAR_TOP指示Android在回退栈中寻找指定的activity实例。如果实例存在，则弹出栈内所有其他activity，让启动的目标activity出现在栈顶（显示在屏幕上），如图13-12所示。

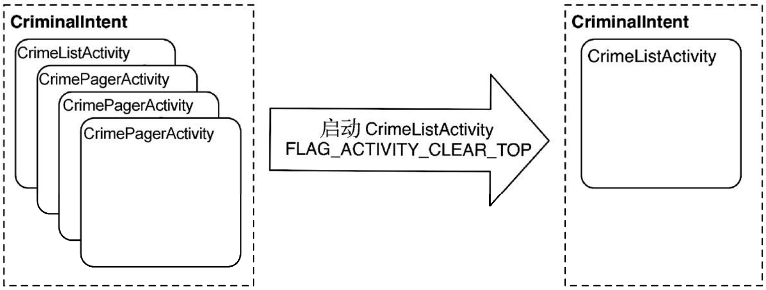


图13-12　工作中的FLAG_ACTIVITY_CLEAR_TOP

13.4　可选菜单项

在本节中，我们将利用前面学过的菜单资源相关知识，添加一个菜单项来实现显示或隐藏CrimeListActivity工具栏的子标题（用来显示crime记录条数）。

打开res/menu/fragment_crime_list.xml文件，参照代码清单13-12，新增一个名为SHOW SUBTITLE的菜单项。如果显示空间足够，它将显示在工具栏上。

代码清单13-12　添加SHOW SUBTITLE菜单项（res/menu/fragment_crime_list.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>

    <item
        android:id="@+id/show_subtitle"
        android:title="@string/show_subtitle"
        app:showAsAction="ifRoom"/>
</menu>
```

子标题需显示crime记录条数，参照代码清单13-13，创建新方法updateSubtitle()实现这个需求。

代码清单13-13 设置工具栏子标题（CrimeListFragment.java）

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);

    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
    ...
```

getString(int resId, Object...formatArgs)方法接受字符串资源中占位符的替换值，updateSubtitle()用它生成子标题字符串。

接着，托管CrimeListFragment的activity被强制类型转换为AppCompatActivity。既然CriminalIntent应用使用了AppCompat库，所有activity就都是AppCompatActivity的子类，自然也能访问工具栏。（由于兼容性问题，在AppCompat库中，工具栏在很多地方仍被称为操作栏。）

在onOptionsItemSelected(...)方法中，调用updateSubtitle()方法响应新增菜单项的单击事件，如代码清单13-14所示。

代码清单13-14 响应SHOW SUBTITLE菜单项单击事件（CrimeListFragment.java）

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            ...
            return true;
        case R.id.show_subtitle:
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

运行CriminalIntent应用，点击SHOW SUBTITLE菜单项，确认子标题显示出crime记录条数。

13.4.1 切换菜单项标题

工具栏上的子标题显示后，菜单项标题依然显示为SHOW SUBTITLE。显然，菜单项标题的

切换与子标题的显示或隐藏需要联动。

调用onOptionsItemSelected(MenuItem)方法时，传入的参数是用户点击的MenuItem。虽然可以在这个方法里更新SHOW SUBTITLE菜单项的文字，但设备旋转并重建工具栏时，子标题的变化会丢失。

比较好的解决方法是在onCreateOptionsMenu(...)方法内更新SHOW SUBTITLE菜单项，并在用户点击子标题菜单项时重建工具栏。对于用户选择菜单项或重建工具栏的场景，都可以使用这段菜单项更新代码。

首先新增跟踪记录子标题状态的成员变量，如代码清单13-15所示。

代码清单13-15 记录子标题状态（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;
    private boolean mSubtitleVisible;
    ...
```

接着，用户点击SHOW SUBTITLE菜单项时，在onCreateOptionsMenu(...)方法内更新子标题，同时重建菜单项，如代码清单13-16所示。

代码清单13-16 更新菜单项（CrimeListFragment.java）

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);

    MenuItem subtitleItem = menu.findItem(R.id.show_subtitle);
    if (mSubtitleVisible) {
        subtitleItem.setTitle(R.string.hide_subtitle);
    } else {
        subtitleItem.setTitle(R.string.show_subtitle);
    }
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            ...
        case R.id.show_subtitle:
            mSubtitleVisible = !mSubtitleVisible;
            getActivity().invalidateOptionsMenu();
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

最后，根据mSubtitleVisible变量值，联动菜单项标题与子标题，如代码清单13-17所示。

代码清单13-17　实现菜单项标题与子标题的联动（CrimeListFragment.java）

```
private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);

    if (!mSubtitleVisible) {
        subtitle = null;
    }

    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
```

运行CriminalIntent应用，确认菜单项标题与子标题能够联动。

13.4.2　“还有个问题”

解决Android编程问题如同对付神探可伦坡[①]的盘问。你以为你的解决方案无懈可击，可以高枕无忧了，但Android每次都会堵在门口提醒道：“还有个问题没解决。”

准确地说，还有两个问题。首先，新建crime记录后，使用后退键回到CrimeListActivity界面，子标题显示的总记录数不会更新。其次，子标题显示后，旋转设备，显示的子标题会消失。

先看记录刷新问题。在返回CrimeListActivity界面时，再次刷新子标题显示就能解决这个问题。也就是说，在onResume方法里再次调用updateSubtitle方法。既然onResume方法和onCreateView方法会调用updateUI方法，那就在updateUI方法里直接调用updateSubtitle方法，如代码清单13-18所示。

代码清单13-18　显示最新状态（CrimeListFragment.java）

```
private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.notifyDataSetChanged();
    }

    updateSubtitle();
}
```

① 美国经典电视电影系列《神探可伦坡》的主角。“还有个问题”（Just one more thing...）是他在破案过程中常说的一句话。——编者注

运行CriminalIntent应用。显示子标题，然后新建crime记录并按后退键返回到CrimeList-Activity界面。可以看到，工具栏显示的总记录数没问题了。

现在，重复上述步骤，但这次改用向上按钮。注意看，子标题显示被重置了！这又是什么情况？

这是Android实现层级式导航带来的问题：导航回退到的目标activity会被完全重建。既然父activity是全新的，实例变量值以及保存的实例状态显然会彻底丢失。

在向上导航时保证子标题的可见状态并不容易。一种方案是覆盖向上导航的机制。在CriminalIntent应用中，调用CrimePagerActivity的finish()方法直接回退到前一个activity界面。遗憾的是，这种方法只能回退一个层级，而实际开发的应用绝大多数都需要多层级导航。

另一种方案是在启动CrimePagerActivity时，把子标题状态作为extra信息传给它。然后，在CrimePagerActivity中覆盖getParentActivityIntent()方法，用附带了extra信息的intent重建CrimeListActivity。这需要CrimePagerActivity类知道父类工作机制的细节。

上述两种方案都不够理想，但目前没有更好的方法。

无论如何，CriminalIntent应用的子标题显示问题算是解决了。现在解决设备旋转问题。只要利用实例状态保存机制，保存mSubtitleVisible实例变量值就能解决问题，如代码清单13-19所示。

代码清单13-19　保存子标题状态值（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {

    private static final String SAVED_SUBTITLE_VISIBLE = "subtitle";
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        if (savedInstanceState != null) {
            mSubtitleVisible = savedInstanceState.getBoolean(SAVED_SUBTITLE_VISIBLE);
        }

        updateUI();

        return view;
    }

    @Override
    public void onResume() {
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
    }
}
```

运行CriminalIntent应用。显示子标题并旋转设备，可以看到子标题在新建视图中依然能正确显示。

13.5　深入学习：工具栏与操作栏

工具栏和操作栏究竟有什么区别呢？



图13-13　工具栏（右）与操作栏（左）

首先，两者给我们最直观的印象就是工具栏界面更美观。工具栏的左边不再放置图标，右边菜单项的间距也更小。另外就是向上的导航按钮。操作栏上的这个按钮不够醒目，只是旁边按钮的附属物。

除了视觉上的差异，在使用上，工具栏比操作栏更灵活。操作栏的使用限制多多，比如，整个应用只能配置一个操作栏且位置及尺寸必须固定（在屏幕顶部）。工具栏就没有这些限制。

本章使用的工具栏应用了AppCompat主题。如果有需要，也可以通过activity和fragment布局定制标准视图的工具栏。可以在屏幕的任何位置摆放工具栏，甚至可以在同一屏配置多个工具栏。应用设计的自由度由此大大提高了，例如，可以为每个fragment定制专用工具栏。可以想象，在同一个用户界面托管多个fragment时，每个fragment都由自己的工具栏控制，这比所有fragment共享一个位于屏幕顶部的工具栏方便多了。

此外，工具栏还能支持内嵌视图和调整高度，这极大地丰富了应用的交互模式。毫不夸张地说，应用设计最大的局限就是我们的想象空间的局限。

13.6　挑战练习：删除 crime 记录

CriminalIntent应用目前不支持删除现有crime记录。请为`CrimeFragment`添加菜单项，允许用户删除当前crime记录。用户点击删除菜单项后，记得调用`CrimeFragment`托管活动的`finish()`方法回退到前一个activity界面。

13.7　挑战练习：复数字符串资源

只有一条crime记录的时候，显示总记录数的子标题会显示：1 crimes。单词crime仍用了复数形式。请改正这个粗心的语法错误。

实现思路上，你可以在代码中准备不同的字符串资源并分情况使用，但这会给应用本地化制造麻烦。比较好的做法是使用复数字符串资源（又称为量化字符串）。

首先，在strings.xml文件中定义复数字符串资源。

```
<plurals name="subtitle_plural">
    <item quantity="one">%1$d crime</item>
    <item quantity="other">%1$d crimes</item>
</plurals>
```

然后，使用getQuantityString方法正确处理单复数问题。

```
int crimeSize = crimeLab.getCrimes().size();
String subtitle = getResources()
    .getQuantityString(R.plurals.subtitle_plural, crimeSize, crimeSize);
```

13.8 挑战练习：用于 RecyclerView 的空视图

当前，CriminalIntent应用启动后，会显示一个空白列表。从用户体验上来讲，即使crime列表是空的，也应展示提示或解释类信息。

请设置空视图并展示类似“没有crime记录可以显示”的信息。再添加一个按钮，方便用户直接创建新的crime记录。

判断crime列表是否包含数据，然后使用任何类都有的setVisibility方法控制占位视图的显示。

第 14 章

SQLite数据库

几乎所有应用都有持久化保存数据的需要。临时性存储`savedInstanceState`显然无法胜任。为此，Android提供了长期存储地：手机或平板设备闪存上的本地文件系统。

Android设备上的应用都有一个沙盒目录。将文件保存在沙盒中，可阻止其他应用甚至是设备用户的访问和窥探。（当然，如果设备被root了的话，用户就可以为所欲为。）

应用的沙盒目录是/data/data/[应用的包名称]。例如，CriminalIntent应用的沙盒目录是/data/data/com.bignerdranch.android.criminalintent。

需要保存大量数据时，大多数应用都不会使用类似txt这样的普通文件。原因很简单：假设将crime记录写入了这样的文件，在仅需要修改crime标题时，就得首先读取整个文件的内容，完成修改后再全部保存。数据量大的话，这将非常耗时。

怎么办呢？有请SQLite登场。SQLite是类似于MySQL和PostgreSQL的开源关系型数据库。不同于其他数据库的是，SQLite使用单个文件存储数据，读写数据时使用SQLite库。Android标准库包含SQLite库以及配套的一些Java辅助类。

本章，我们仅学习如何运用SQLite基本辅助类，打开应用沙盒中的数据库，读取或写入数据。若需深入学习，请访问网站www.sqlite.org，阅读SQLite完全使用手册。

14.1　定义 schema

创建数据库前，首先要清楚存储什么样的数据。CriminalIntent应用要保存的是一条条crime记录，这需要定义如图14-1所示的`crimes`数据表。

_id	uuid	title	date	solved
1	13090636733242	Stolen yogurt	13090636733242	0
2	13090732131909	Dirty sink	13090732131909	1

图14-1　`crimes`数据表

定义schema的方式众多，如何选择往往因人而异。但所有方式都有一个共同的目标：“不要重复造轮子。”实际上，这也是人人都应遵守的编程准则：多花时间思考复用代码的编写和调用，避免在应用中到处使用重复代码。

使用数据库也是如此。面对众多工具，你甚至可以用上能统一定义模型层对象（如Crime）的高级ORM（对象关系映射）。不过，简单起见，对于CriminalIntent应用，我们打算直接在Java代码中定义database schema（描述表名和数据字段）。

首先创建定义schema的Java类。将其命名为CrimeDbSchema，同时在新建类对话框中输入包名database.CrimeDbSchema。这样，就可以将CrimeDbSchema.java文件放入专门的database包中，实现数据库相关代码的统一组织和归类。

在CrimeDbSchema类中，再定义一个描述数据表的CrimeTable内部类，如代码清单14-1所示。

代码清单14-1 定义CrimeTable内部类（CrimeDbSchema.java）

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";
    }
}
```

CrimeTable内部类唯一的用途就是定义描述数据表元素的String常量。它的首个定义是数据库表名（CrimeTable.NAME）。

接下来定义数据表字段，如代码清单14-2所示。

代码清单14-2 定义数据表字段（CrimeDbSchema.java）

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
        }
    }
}
```

有了这些数据表元素，就可以在Java代码中安全地引用了。例如，CrimeTable.Cols.TITLE就是指crime记录的title字段。此外，这还给修改字段名称或新增表元素带来了方便。

14

14.2 创建初始数据库

有了数据库schema，就可以创建数据库了。openOrCreateDatabase(...)和databaseList()是Android提供的Context底层方法，用来打开数据库文件并将其转换为SQLiteDatabase实例。

不过，实践中，建议总是遵循以下步骤。

(1) 确认目标数据库是否存在。

(2) 如果不存在，首先创建数据库，然后创建数据表并初始化数据。

(3) 如果存在，打开并确认CrimeDbSchema是否为最新（CriminalIntent后续版本可能有改动）。

(4) 如果是旧版本，就先升级到最新版本。

以上工作可借助Android的SQLiteOpenHelper类处理。在数据库包中创建CrimeBaseHelper类，如代码清单14-3所示。

代码清单14-3　创建CrimeBaseHelper类（CrimeBaseHelper.java）

```
public class CrimeBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "crimeBase.db";

    public CrimeBaseHelper(Context context) {
        super(context, DATABASE_NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
}
```

有了SQLiteOpenHelper类，打开SQLiteDatabase的繁杂工作就简单多了。在CrimeLab中用它创建crime数据库，如代码清单14-4所示。

代码清单14-4　打开SQLiteDatabase（CrimeLab.java）

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
    private Context mContext;
    private SQLiteDatabase mDatabase;
    ...
    private CrimeLab(Context context) {
      mContext = context.getApplicationContext();
      mDatabase = new CrimeBaseHelper(mContext)
              .getWritableDatabase();
      mCrimes = new ArrayList<>();
    }
```

（你可能会问，为什么要把context赋值给实例变量呢？不要奇怪，CrimeLab会在第16章用到它。）

调用getWritableDatabase()方法时，CrimeBaseHelper会做如下工作。

(1) 打开/data/data/com.bignerdranch.android.criminalintent/databases/crimeBase.db数据库；如果不存在，就先创建crimeBase.db数据库文件。

(2) 如果是首次创建数据库，就调用`onCreate(SQLiteDatabase)`方法，然后保存最新的版本号。

(3) 如果已创建过数据库，首先检查它的版本号。如果`CrimeBaseHelper`中的版本号更高，就调用`onUpgrade(SQLiteDatabase, int, int)`方法升级。

最后做个总结：`onCreate(SQLiteDatabase)`方法负责创建初始数据库；`onUpgrade-(SQLiteDatabase, int, int)`方法负责与升级相关的工作。

CriminalIntent当前只有一个版本，暂时可以忽略`onUpgrade(...)`方法。目前，只需要在`onCreate(...)`方法中创建数据表。这需要导入`CrimeDbSchema`类的`CrimeTable`内部类。

导入分两步完成。首先，编写SQL创建初始代码，如代码清单14-5所示。

代码清单14-5　编写SQL创建初始代码（CrimeBaseHelper.java）

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeDbSchema.CrimeTable.NAME);
}
```

把光标移到`CrimeTable`一词上，按Option+Return（或Alt+Enter）组合键，然后选择提示里的Add import for 'com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable'可选项，如图14-2所示。

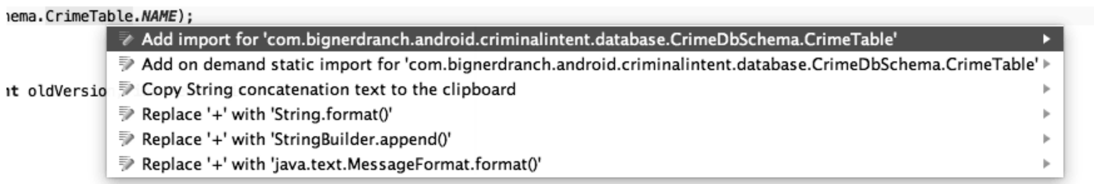


图14-2　导入CrimeTable

Android Studio会自动产生如下导入语句：

```
import com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable;

public class CrimeBaseHelper extends SQLiteOpenHelper {
```


这样，与其费事地输入`CrimeDbSchema.CrimeTable.Cols.UUID`，还不如直接以`CrimeTable.Cols.UUID`的形式使用`CrimeDbSchema.CrimeTable`中的`String`常量。完成其余表定义代码，如代码清单14-6所示。

代码清单14-6　创建crime数据表（CrimeBaseHelper.java）

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeTable.NAME + "(" +
            " _id integer primary key autoincrement, " +
            CrimeTable.Cols.UUID + ", " +
```

```
                CrimeTable.Cols.TITLE + ", " +
                CrimeTable.Cols.DATE + ", " +
                CrimeTable.Cols.SOLVED +
                ")"
        );
    }
```

相比其他数据库，创建SQLite数据库表省事又简单：创建表字段时，不需要指定表字段类型。

运行CriminalIntent应用后，数据库将被创建。如果在用模拟器或是已root的设备，可直接看到已创建的数据库文件。

写作本书时，Nougat（API 24或25）模拟器镜像无法配合Android Device Monitor文件浏览器使用。要看到刚创建的文件，你需要使用旧版本的模拟器安装并运行应用。

14.2.1　使用 Android Device Monitor 查看文件

改用旧版本模拟器后，请选择Tools → Android → Android Device Monitor菜单项，如果看到要求禁用ADB整合的对话框，请选Yes，如图14-3所示。



图14-3　禁用ADB整合

等Android Device Monitor窗口弹出后，选择File Explorer选项页。浏览至/data/data/com.bignerdranch.android.criminalintent/databases/目录，即可看到CriminalIntent刚创建的数据库文件，如图14-4所示。

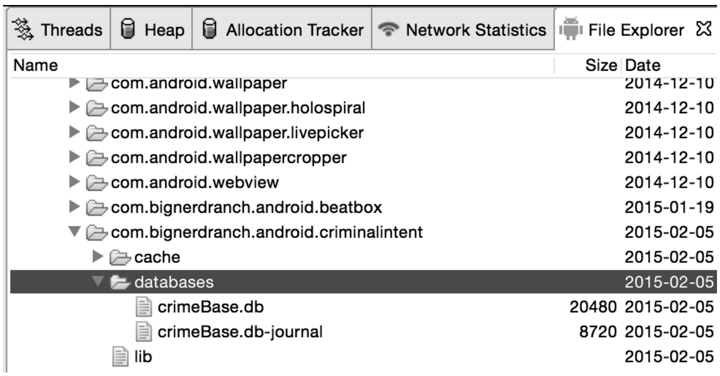


图14-4　数据库文件已生成

（禁用ADB整合后，如果运行应用，你会看到Instant Run requires 'Tools | Android | Enable ADB integration' to be enabled这样的错误提示。要解决这个问题，请选择Android Studio → Preferences菜单项，在弹出的首选项界面的左上角输入“Instant Run”搜索，如图14-5所示。然后，去除Enable Instant Run to hot swap code/resource changes on deploy (default enabled)选项，点选Apply按钮应用并点OK按钮完成。）

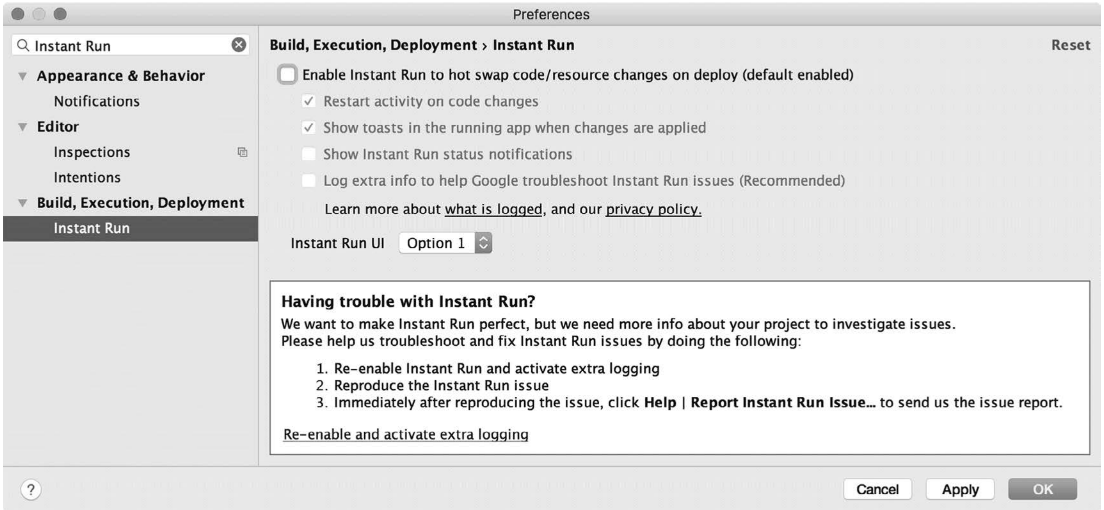


图14-5　禁用Instant Run

14.2.2　处理数据库相关问题

编写SQLite数据库操作代码时，经常会碰到要调整数据库表结构的情况。例如，我们还要在后续章节中添加crime嫌疑人。这就需要为crime数据表新增字段。处理时，比较常规的做法是，在`SQLiteOpenHelper`中记录版本号，然后在`onUpgrade(...)`方法中升级数据表。

这种常规方法涉及不少代码量。而且，编写和维护很少更新版本的代码也很伤脑筋。所以，实践中，最好的做法是直接删除数据库文件，再让`SQLiteOpenHelper.onCreate(...)`方法重建。

直接从设备上删除应用是删除数据库文件最便捷的方法。要在模拟器上删除应用，可以切换到应用浏览器界面，向上拖动CriminalIntent应用的图标，直到屏幕顶部出现Uninstall字样（注意，不同Android版本的操作可能有所差别）。执行并确认删除应用，如图14-6所示。

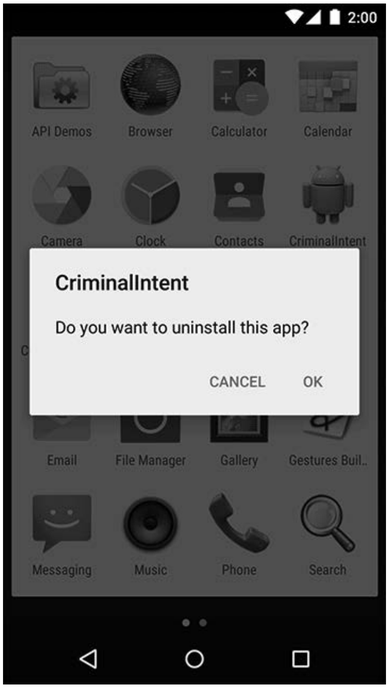


图14-6　删除应用

记住，在学习本章的过程中，如遇数据库相关问题，都建议直接删除应用，从头再来。

14.3　修改 CrimeLab 类

创建完数据库，接下来是调整CrimeLab类代码，改用mDatabase存储数据。

首先要删除CrimeLab中所有与mCrimes相关的代码，如代码清单14-7所示。

代码清单14-7　删除mCrimes相关代码（CrimeLab.java）

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
    private Context mContext;
    private SQLiteDatabase mDatabase;

    public static CrimeLab get(Context context) {
        ...
    }

    private CrimeLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new CrimeBaseHelper(mContext)
```

```
                .getWritableDatabase();
        mCrimes = new ArrayList<>();
    }

    public void addCrime(Crime c) {
        mCrimes.add(c);
    }

    public List<Crime> getCrimes() {
        return mCrimes;
        return new ArrayList<>();
    }

    public Crime getCrime(UUID id) {
        for (Crime crime : mCrimes) {
            if (crime.getId().equals(id)) {
                return crime;
            }
        }
        return null;
    }
}
```

代码调整完毕。运行CriminalIntent应用时，只会看到空列表和空CrimePagerActivity。没关系，下面就来逐步完善它。

14.4　写入数据库

要使用SQLiteDatabase，首先要有数据。数据库写入操作有：向crime表中插入新记录以及在Crime变更时更新原始记录。

14.4.1　使用 ContentValues

负责处理数据库写入和更新操作的辅助类是ContentValues。它是一个键值存储类，类似于Java的HashMap和前面用过的Bundle。不同的是，ContentValues只能用于处理SQLite数据。

将Crime记录转换为ContentValues，实际就是在CrimeLab中创建ContentValues实例。这需要新建一个私有方法，如代码清单14-8所示。（记得使用前述的两个步骤导入CrimeTable：把光标定位到CrimeTable.Cols.UUID上，按Option+Return（或Alt+Enter）组合键，然后选择提示里的Add import for 'com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable'可选项。）

代码清单14-8　创建ContentValues（CrimeLab.java）

```
    public Crime getCrime(UUID id) {
        return null;
    }

    private static ContentValues getContentValues(Crime crime) {
```

```
        ContentValues values = new ContentValues();
        values.put(CrimeTable.Cols.UUID, crime.getId().toString());
        values.put(CrimeTable.Cols.TITLE, crime.getTitle());
        values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
        values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);

        return values;
    }
}
```

ContentValues的键就是数据表字段。注意不要搞错，否则会导致数据插入和更新失败。除了_id是由数据库自动创建外，其他所有数据表字段都要编码指定。

14.4.2　插入和更新记录

准备好ContentValues，就该向数据库写入数据了。参照代码清单14-9，在addCrime(Crime)方法中新增数据插入实现代码。

代码清单14-9　插入记录（CrimeLab.java）

```
public void addCrime(Crime c) {
    ContentValues values = getContentValues(c);

    mDatabase.insert(CrimeTable.NAME, null, values);
}
```

insert(String, String, ContentValues)方法的第一和第三个参数很重要，第二个很少用到。传入的第一个参数是数据表名（CrimeTable.NAME），第三个是要写入的数据。

第二个参数称为nullColumnHack。它有什么用途呢?

别急，举个例子你就明白了。假设你想调用insert(...)方法，但传入了ContentValues空值。这时，SQLite不干了，insert(...)方法调用只能以失败告终。

然而，如果能以uuid值作为nullColumnHack传入的话，SQLite就可以忽略ContentValues空值，而且还会自动传入一个带uuid且值为null的ContentValues。结果，insert(...)方法得以成功调用并插入了一条新记录。

听起来不错吧，也许某天会用得着，但肯定不是现在。因此，目前只要了解就可以了。

完成了数据插入，下面继续使用ContentValues，新增数据库记录更新方法，如代码清单14-10所示。

代码清单14-10　更新记录（CrimeLab.java）

```
public Crime getCrime(UUID id) {
    return null;
}

public void updateCrime(Crime crime) {
    String uuidString = crime.getId().toString();
    ContentValues values = getContentValues(crime);
```

```
        mDatabase.update(CrimeTable.NAME, values,
                CrimeTable.Cols.UUID + " = ?",
                new String[] { uuidString });
    }

    private static ContentValues getContentValues(Crime crime) {
```

update(String, ContentValues, String, String[])方法类似于insert(...)方法，向其传入要更新的数据表名和为表记录准备的ContentValues。然而，与insert(...)方法不同的是，你要确定该更新哪些记录。具体的做法是：创建where子句（第三个参数），然后指定where子句中的参数值（String[]数组参数）。

问题来了，为什么不直接在where子句中放入uuidString呢？这可比使用?然后传入String[]简单多了！

事实上，很多时候，String本身会包含SQL代码。如果将它直接放入query语句中，这些代码可能会改变query语句的含义，甚至会修改数据库资料。这实际就是SQL脚本注入，其危害相当严重。

使用?的话，就不用关心String包含什么，代码执行的效果肯定就是我们想要的。因此，建议你保持这种良好的编码习惯。

用户可能会在CrimeFragment中修改Crime实例。修改完成后，你需要刷新CrimeLab中的Crime数据。这可以通过覆盖CrimeFragment.onPause()方法完成，如代码清单14-11所示。

代码清单14-11　Crime数据刷新（CrimeFragment.java）

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    }

    @Override
    public void onPause() {
        super.onPause();

        CrimeLab.get(getActivity())
                .updateCrime(mCrime);
    }
```

数据库写入部分处理完了。不过，要检验成果，得等到数据库读取也处理完。继续之前，运行CriminalIntent应用看看代码能否正常编译。一切正常的话，你应该看到一个crime空列表。

14.5　读取数据库

读取数据需要用到SQLiteDatabase.query(...)方法。这个方法有好几个重载版本。我们要用的版本如下：

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

以前写过SQL代码的话，你应该已经熟悉这些`select`语句参数了。不熟悉也没关系，只关注眼前要用的就好了。

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

参数`table`是要查询的数据表。参数`columns`指定要依次获取哪些字段的值。参数`where`和`whereArgs`的作用与`update(...)`方法中的一样。

新增一个便利方法，调用`query(...)`方法查询`CrimeTable`中的记录，如代码清单14-12所示。

代码清单14-12　查询crime记录（CrimeLab.java）

```
public void updateCrime(Crime crime) {
    ...
}

private Cursor queryCrimes(String whereClause, String[] whereArgs) {
    Cursor cursor = mDatabase.query(
            CrimeTable.NAME,
            null, // Columns - null selects all columns
            whereClause,
            whereArgs,
            null, // groupBy
            null, // having
            null  // orderBy
    );

    return cursor;
}
```

14.5.1　使用 CursorWrapper

`Cursor`是个神奇的表数据处理工具。其功能就是封装数据表中的原始字段值。从`Cursor`获取数据的代码大致如下：

```
String uuidString = cursor.getString(
    cursor.getColumnIndex(CrimeTable.Cols.UUID));
String title = cursor.getString(
    cursor.getColumnIndex(CrimeTable.Cols.TITLE));
long date = cursor.getLong(
    cursor.getColumnIndex(CrimeTable.Cols.DATE));
int isSolved = cursor.getInt(
    cursor.getColumnIndex(CrimeTable.Cols.SOLVED));
```

每从Cursor中取出一条crime记录，以上代码都要重复写一次。（这还不包括按照这些字段值创建Crime实例的代码。）

显然，遇到这种情况，首先要考虑代码复用。与其机械地编写重复代码，不如创建可复用的专用Cursor子类。创建Cursor子类最简单的方式是使用CursorWrapper。顾名思义，你可以用CursorWrapper封装Cursor的对象，然后再添加有用的扩展方法。

在数据库包中新建CrimeCursorWrapper类，如代码清单14-13所示。

代码清单14-13　创建CrimeCursorWrapper类（CrimeCursorWrapper.java）

```
public class CrimeCursorWrapper extends CursorWrapper {
    public CrimeCursorWrapper(Cursor cursor) {
        super(cursor);
    }
}
```

以上代码创建了一个Cursor封装类。该类继承了Cursor类的全部方法。注意，这样封装的目的就是定制新方法，以方便操作内部Cursor。

参照代码清单14-14，新增获取相关字段值的getCrime()方法。（别忘了前面介绍的两步导入CrimeTable的技巧。）

代码清单14-14　新增getCrime()方法（CrimeCursorWrapper.java）

```
public class CrimeCursorWrapper extends CursorWrapper {
    public CrimeCursorWrapper(Cursor cursor) {
        super(cursor);
    }

    public Crime getCrime() {
        String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
        String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
        long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
        int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

        return null;
    }
}
```

我们需要返回具有UUID的Crime。在Crime.java中新增一个有此用途的构造方法，如代码清单14-15所示。

14

代码清单14-15　新增Crime构造方法（Crime.java）

```
public Crime() {
    this(UUID.randomUUID());
    mId = UUID.randomUUID();
    mDate = new Date();
}

public Crime(UUID id) {
    mId = id;
    mDate = new Date();
}
```

最后，完成getCrime()方法，如代码清单14-16所示。

代码清单14-16　完成getCrime()方法（CrimeCursorWrapper.java）

```
public Crime getCrime() {
    String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
    String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
    long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
    int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

    Crime crime = new Crime(UUID.fromString(uuidString));
    crime.setTitle(title);
    crime.setDate(new Date(date));
    crime.setSolved(isSolved != 0);

    return crime;
    return null;
}
```

（Android Studio会询问是选择java.util.Date还是java.sql.Date。不要搞错，即便现在是在编写数据库相关代码，也应该选java.util.Date。）

14.5.2　创建模型层对象

使用CrimeCursorWrapper类，可直接从CrimeLab中取得List<Crime>。大致思路无外乎将查询返回的cursor封装到CrimeCursorWrapper类中，然后调用getCrime()方法遍历取出Crime。

首先，让queryCrimes(...)方法返回CrimeCursorWrapper对象，如代码清单14-17所示。

代码清单14-17　使用Cursor封装方法（CrimeLab.java）

```
private Cursor queryCrimes(String whereClause, String[] whereArgs) {
private CrimeCursorWrapper queryCrimes(String whereClause, String[] whereArgs) {
    Cursor cursor = mDatabase.query(
            CrimeTable.NAME,
            null, // Columns - null selects all columns
            whereClause,
            whereArgs,
            null, // groupBy
            null, // having
```

```
            null  // orderBy
        );

        return cursor;
        return new CrimeCursorWrapper(cursor);
    }
```

然后，完善getCrime()方法：遍历查询取出所有的crime，返回Crime数组对象，如代码清单14-18所示。

代码清单14-18　返回crime列表（CrimeLab.java）

```
    public List<Crime> getCrimes() {
        return new ArrayList<>();
        List<Crime> crimes = new ArrayList<>();

        CrimeCursorWrapper cursor = queryCrimes(null, null);

        try {
            cursor.moveToFirst();
            while (!cursor.isAfterLast()) {
                crimes.add(cursor.getCrime());
                cursor.moveToNext();
            }
        } finally {
            cursor.close();
        }

        return crimes;
    }
```

数据库cursor之所以被称为cursor，是因为它内部就像有根手指似的，总是指向查询的某个地方。因此，要从cursor中取出数据，首先要调用moveToFirst()方法移动虚拟手指指向第一个元素。读取行记录后，再调用moveToNext()方法，读取下一行记录，直到isAfterLast()说没有数据可取为止。

最后，别忘了调用Cursor的close()方法关闭它，否则会出错：轻则应用报错，重则应用崩溃。

CrimeLab.getCrime(UUID)方法类似于getCrimes()方法，唯一区别就是，它只需要取出已存在的首条记录，如代码清单14-19所示。

代码清单14-19　重写getCrime(UUID)方法（CrimeLab.java）

```
    public Crime getCrime(UUID id) {
        return null;
        CrimeCursorWrapper cursor = queryCrimes(
                CrimeTable.Cols.UUID + " = ?",
                new String[] { id.toString() }
        );

        try {
            if (cursor.getCount() == 0) {
                return null;
```

```
            }

            cursor.moveToFirst();
            return cursor.getCrime();
        } finally {
            cursor.close();
        }
    }
```

上述代码的作用总结如下。

- 现在可以插入crime记录了。也就是说，点击New Crime菜单项，将Crime添加到CrimeLab的代码可以工作了。
- 数据库查询没有问题了。CrimePagerActivity现在能够看见CrimeLab中的所有Crime了。
- CrimeLab.getCrime(UUID)方法也能正常工作了。CrimePagerActivity托管的CrimeFragment终于可以显示真正的Crime对象了。

现在，点击New Crime菜单项可以在CrimePagerActivity界面看到新增的Crime了。运行CriminalIntent应用确认。

刷新模型层数据

这还不算完。虽然Crime记录已存入数据库，但数据读取还未完善。例如，编辑完新的crime后，尝试点击后退键，你会发现CrimeListActivity并没有相应刷新。

这是因为CrimeLab的工作方式已经变了。以前，只有一个List<Crime>，而且每个Crime在List<Crime>中只存有一个对象。要获取哪个Crime只能去找mCrimes。

现在，mCrimes已废弃不用。所以，getCrimes()方法返回的List<Crime>是Crime对象的快照。要刷新CrimeListActivity界面，首先要更新这个快照。

好在大部分基础工作已经就绪。CrimeListActivity目前调用updateUI()方法来刷新界面。剩下的事情就是刷新CrimeLab的视图了。

要刷新crime显示，首先添加一个setCrimes(List<Crime>)方法给CrimeAdapter，如代码清单14-20所示。

代码清单14-20　添加setCrimes(List<Crime>)方法（CrimeListFragment.java）

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
    @Override
    public int getItemCount() {
        return mCrimes.size();
    }

    public void setCrimes(List<Crime> crimes) {
        mCrimes = crimes;
    }
}
```

然后在updateUI()方法中调用setCrimes(List<Crime>)方法，如代码清单14-21所示。

代码清单14-21 调用setCrimes(List<Crime>)方法（CrimeListFragment.java）

```
private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.setCrimes(crimes);
        mAdapter.notifyDataSetChanged();
    }

    updateSubtitle();
}
```

现在，可以验证我们的成果了。运行CriminalIntent应用，新增一项crime记录，然后按后退键，确认CrimeListActivity中会出现刚才新增的记录。

此外，还建议验证CrimeFragment中的updateCrime(Crime)方法的调用是否正常。点击任意crime列表项，在CrimePagerActivity中编辑它的标题。然后按后退键，确认对应列表项的标题会更新。

14.6 深入学习：数据库高级主题介绍

为了简化教学及控制篇幅，本章没有讨论数据库应用的方方面面。要知道，很多专业应用都需要高级数据库使用支持。为了提高开发效率，人们常常会求助于ORM这样专业又复杂的工具。

开发复杂的数据库应用时，难免会需要以下数据库元素。

- *字段类型*。从技术实现上讲，SQLite的字段不需要指定数据类型。没有它们完全不影响数据库的使用；当然，如果能给出数据类型提示会更好。
- *索引*。查询数据库字段时，字段不加索引会严重影响查询性能和效率。
- *外键*。本章的数据库只用了一张表。如果涉及多张表，关联数据应该需要外键约束。

数据库性能优化需要考虑的因素众多。每次查询数据库时，CriminalIntent应用都会创建包含所有Crime对象的列表。要获得高性能表现，需要采取一些优化措施。比如，循环使用Crime实例，或者把它们当作内存对象。显然，这涉及更多的代码量。当然，这也是ORM这样的专业工具要解决的问题。

14

14.7 深入学习：应用上下文

本章一开始，我们在CrimeLab的构造方法中使用了*应用上下文*。

```
private CrimeLab(Context context) {
  mContext = context.getApplicationContext();
  ...
}
```

应用上下文有什么特别呢？就上例来看，为什么要用应用上下文，而不直接用activity作为context呢？

要回答上述问题，关键就在于考虑它们的生命周期。只要有activity在，Android肯定也创建了application对象。用户在应用的不同界面间导航时，各个activity时而存在时而消亡，但application对象不会受任何影响。可以说，它的生命周期要比任何activity都长。

CrimeLab是一个单例。这表明，一旦创建，它就会一直存在，直至整个应用进程被销毁。由代码可知，CrimeLab引用着mContext对象。显然，如果把activity作为mContext对象保存的话，这个由CrimeLab一直引用着的activity肯定会免遭垃圾回收器的清理，即便用户跳转离开这个activity时也是如此。

为了避免资源浪费，我们使用了应用上下文。这样，CrimeLab仍可以引用Context对象，而activity的生死也不用受它束缚了。

14.8　挑战练习：删除 crime 记录

如果为应用添加过Delete Crime菜单项的话，就可以直接调用CrimeLab的deleteCrime-(Crime)方法，继而调用mDatabase.delete(...)方法来实现删除功能。

如果还没有，那就先给CrimeFragment的工具栏添加一个Delete Crime菜单项，然后调用CrimeLab.deleteCrime(Crime)方法实现删除功能。

第 15 章

隐式intent

15

在Android系统中，可利用隐式intent启动其他应用的activity。在显式intent中，我们指定要启动的activity类，操作系统会负责启动它。在隐式intent中，我们只要描述要完成的任务，操作系统就会找到合适的应用，并在其中启动相应的activity。

本章，我们将使用隐式intent发送短消息给Crime嫌疑人。用户首先从某个联系人应用中选取联系人，然后从短消息应用列表中选取目标应用发送消息，如图15-1所示。



图15-1　打开联系人应用和消息发送应用

对于开发者来说，使用隐式intent利用其他应用完成常见任务，远比自己编写代码从头实现要容易得多。对于用户来说，他们也乐意在应用中调用自己熟悉或喜爱的应用。

创建隐式intent之前，需完成以下准备工作：

- 在`CrimeFragment`的布局上添加CHOOSE SUSPECT按钮和SEND CRIME REPORT按钮；
- 在`Crime`类中添加保存嫌疑人姓名的`mSuspect`变量；
- 使用格式化的字符串资源创建消息模板。

15.1　添加按钮组件

首先，我们在CrimeFragment布局中添加两个投诉用按钮：一个嫌疑人选取按钮（CHOOSE SUSPECT按钮）和一个消息发送按钮（SEND CRIME REPORT按钮）。添加按钮前，先来添加显示在按钮上的字符串资源，如代码清单15-1所示。

代码清单15-1　添加按钮用字符串（strings.xml）

```
        <string name="subtitle_format">%1$d crimes</string>
        <string name="crime_suspect_text">Choose Suspect</string>
        <string name="crime_report_text">Send Crime Report</string>
    </resources>
```

然后，在layout/fragment_crime.xml布局文件中，参照图15-2，添加两个按钮组件。注意，为了突出重点，我们在布局定义示意图中隐藏了LinearLayout的部分子元素。

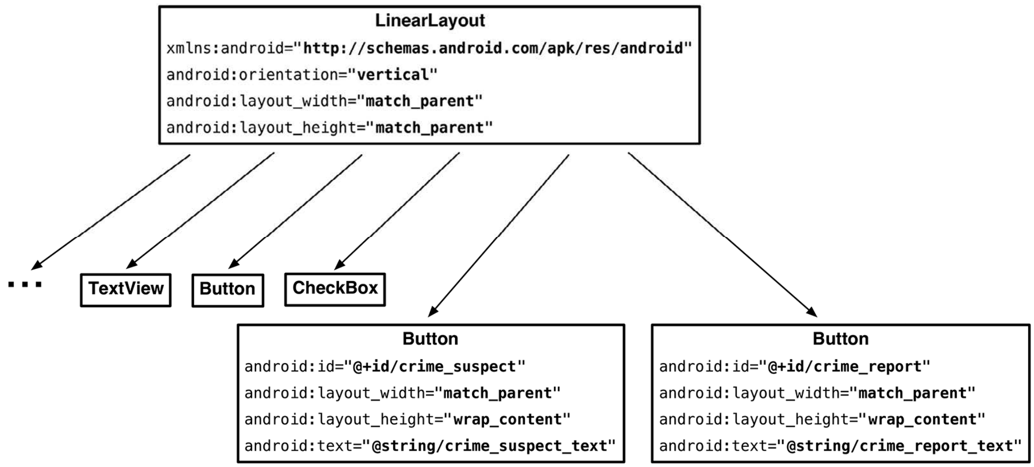


图15-2　添加嫌疑人选取按钮和消息发送按钮（layout/fragment_crime.xml）

现在，可以预览新布局了。当然，也可以直接运行CriminalIntent应用，确认新增加的按钮显示正常。

15.2　添加嫌疑人信息至模型层

接下来，返回到Crime.java中，新增存储嫌疑人姓名的mSuspect成员变量，如代码清单15-2所示。

代码清单15-2　添加mSuspect成员变量（Crime.java）

```
public class Crime {
    ...
    private boolean mSolved;
```

```
    private String mSuspect;

    public Crime() {
        this(UUID.randomUUID());
    }
    ...
    public void setSolved(boolean solved) {
        mSolved = solved;
    }

    public String getSuspect() {
        return mSuspect;
    }

    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }
}
```

现在，需要新增crime数据库字段。首先，在数据库schema中定义嫌疑人字段，如代码清单15-3所示。

代码清单15-3　添加嫌疑人数据库字段（CrimeDbSchema.java）

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
            public static final String SUSPECT = "suspect";
        }
    }
}
```

同时，也要在CrimeBaseHelper中新增嫌疑人数据库字段，如代码清单15-4所示。（注意，别忘了CrimeTable.Cols.SOLVED后的逗号。）

代码清单15-4　添加嫌疑人数据库字段（CrimeBaseHelper.java）

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeTable.NAME + "(" +
            " _id integer primary key autoincrement, " +
            CrimeTable.Cols.UUID + ", " +
            CrimeTable.Cols.TITLE + ", " +
            CrimeTable.Cols.DATE + ", " +
            CrimeTable.Cols.SOLVED + ", " +
            CrimeTable.Cols.SUSPECT +
            ")"
    );
}
```

15

接下来，更新`CrimeLab.getContentValues(Crime)`方法中的数据库写入代码，如代码清单15-5所示。

代码清单15-5　写入嫌疑人信息（CrimeLab.java）

```
    private static ContentValues getContentValues(Crime crime) {
        ContentValues values = new ContentValues();
        values.put(CrimeTable.Cols.UUID, crime.getId().toString());
        values.put(CrimeTable.Cols.TITLE, crime.getTitle());
        values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
        values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
        values.put(CrimeTable.Cols.SUSPECT, crime.getSuspect());

        return values;
    }
}
```

最后，更新`CrimeCursorWrapper`中的数据库读取代码，如代码清单15-6所示。

代码清单15-6　读取嫌疑人信息（CrimeCursorWrapper.java）

```
    public Crime getCrime() {
        String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
        String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
        long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
        int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));
        String suspect = getString(getColumnIndex(CrimeTable.Cols.SUSPECT));

        Crime crime = new Crime(UUID.fromString(uuidString));
        crime.setTitle(title);
        crime.setDate(new Date(date));
        crime.setSolved(isSolved != 0);
        crime.setSuspect(suspect);

        return crime;
    }
}
```

注意，如果设备上已安装CriminalIntent应用，当前应用中的数据库是不包含嫌疑人字段的，而且`onCreate(SQLiteDatabase)`方法也不会添加这个新字段。这时，最容易的解决办法就是删除旧数据库（这是开发过程中常有的事）。

如前所述，首先删除CriminalIntent应用。然后，从Android Studio中重新安装并运行它。这样，带有新增字段的数据库就创建成功了。

15.3　使用格式化字符串

最后一项准备工作是创建消息模板。应用运行前，我们无法获知具体的陋习细节。因此，必须使用带有占位符（可在应用运行时替换）的格式化字符串。下面是将要使用的格式化字符串：

```
<string name="crime_report">%1$s! The crime was discovered on %2$s. %3$s, and %4$s
```

`%1$s`、`%2$s`等特殊字符串即为占位符，它们接受字符串参数。在代码中，我们将调用

getString(...)方法，并传入格式化字符串资源ID以及另外四个字符串参数（与要替换的占位符顺序一致）。

首先，在strings.xml中，添加如代码清单15-7所示的字符串资源。

代码清单15-7 添加字符串资源（strings.xml）

```
    <string name="crime_suspect_text">Choose Suspect</string>
    <string name="crime_report_text">Send Crime Report</string>
    <string name="crime_report">%1$s!
      The crime was discovered on %2$s. %3$s, and %4$s
    </string>
    <string name="crime_report_solved">The case is solved</string>
    <string name="crime_report_unsolved">The case is not solved</string>
    <string name="crime_report_no_suspect">there is no suspect.</string>
    <string name="crime_report_suspect">the suspect is %s.</string>
    <string name="crime_report_subject">CriminalIntent Crime Report</string>
    <string name="send_report">Send crime report via</string>
</resources>
```

然后，在CrimeFragment.java中添加getCrimeReport()方法，创建四段字符串信息，并返回拼接完整的消息，如代码清单15-8所示。

代码清单15-8 新增getCrimeReport()方法（CrimeFragment.java）

```
    private void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
    }

    private String getCrimeReport() {
        String solvedString = null;
        if (mCrime.isSolved()) {
            solvedString = getString(R.string.crime_report_solved);
        } else {
            solvedString = getString(R.string.crime_report_unsolved);
        }

        String dateFormat = "EEE, MMM dd";
        String dateString = DateFormat.format(dateFormat, mCrime.getDate()).toString();

        String suspect = mCrime.getSuspect();
        if (suspect == null) {
            suspect = getString(R.string.crime_report_no_suspect);
        } else {
            suspect = getString(R.string.crime_report_suspect, suspect);
        }

        String report = getString(R.string.crime_report,
            mCrime.getTitle(), dateString, solvedString, suspect);

        return report;
    }
}
```

（注意，有两个DateFormat类：android.text.format.DateFormat和java.text.DateFormat。我们要用的是android.text.format.DateFormat。）

至此，准备工作全部完成了，接下来学习如何使用隐式intent。

15.4　使用隐式 intent

Intent对象用来向操作系统说明需要处理的任务。使用显式intent时，我们需指定要操作系统启动哪个activity。下面是之前创建过的显式intent：

```
Intent intent = new Intent(getActivity(), CrimePagerActivity.class);
intent.putExtra(EXTRA_CRIME_ID, crimeId);
startActivity(intent);
```

使用隐式intent时，只需告诉操作系统你想要做什么，操作系统就会去启动能够胜任工作任务的activity。如果找到多个符合的activity，用户会看到一个可选应用列表，然后就看用户如何选择了。

15.4.1　隐式 intent 的组成

下面是隐式intent的主要组成部分，可以用来定义你想做的事。

(1) 要执行的操作

通常以Intent类中的常量来表示。例如，要访问某个URL，可以使用Intent.ACTION_VIEW；要发送邮件，可以使用Intent.ACTION_SEND。

(2) 待访问数据的位置

这可能是设备以外的资源，如某个网页的URL，也可能是指向某个文件的URI，或者是指向ContentProvider中某条记录的某个内容URI（content URI）。

(3) 操作涉及的数据类型

这指的是MIME形式的数据类型，如text/html或audio/mpeg3。如果一个intent包含数据位置，那么通常可以从中推测出数据的类型。

(4) 可选类别

操作用于描述具体要做什么，而类别通常用来描述你打算何时、何地或者如何使用某个activity。例如，Android的android.intent.category.LAUNCHER类别表明，activity应该显示在顶级应用启动器中；而android.intent.category.INFO类别表明，虽然activity向用户显示了包信息，但它不应该出现在启动器中。

一个查看某个网址的简单隐式intent会包括一个Intent.ACTION_VIEW操作，以及某个具体URL网址的Uri数据。

基于以上信息，操作系统将启动适用的activity。（如果有多个应用适用，用户自己挑。）

通过配置文件中的intent过滤器设置，activity会对外宣称自己是适合处理ACTION_VIEW的activity。例如，如果想开发一款浏览器应用，为了响应ACTION_VIEW操作，你会在activity声明中包含以下intent过滤器：

```
<activity
    android:name=".BrowserActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" android:host="www.bignerdranch.com" />
    </intent-filter>
</activity>
```

必须在intent过滤器中明确地设置DEFAULT类别。action元素告诉操作系统，activity能够胜任指定任务。DEFAULT类别告诉操作系统（问谁可以做时），activity愿意处理某项任务。DEFAULT类别实际隐含于所有隐式intent中。（当然也有例外，详见第24章。）

和显式intent一样，隐式intent也可以包含extra信息。不过，操作系统在寻找适用的activity时，不会使用附加在隐式intent上的任何extra。

注意，显式intent也可以使用隐式intent的操作和数据部分。这相当于要求特定的activity去做特定的事。

15.4.2 发送消息

在CriminalIntent应用中，通过创建发送消息的隐式intent，我们来看看它是如何工作的。消息是由字符串组成的文本信息，我们的任务是发送一段文本信息，因此隐式intent的操作是ACTION_SEND。它不指向任何数据，也不包含任何类别，但会指定数据类型为text/plain。

在CrimeFragment.onCreateView(...)方法中，首先以资源ID引用SEND CRIME REPORT按钮并为其设置一个监听器。然后在监听器接口实现中，创建一个隐式intent并传入startActivity(Intent)方法，如代码清单15-9所示。

代码清单15-9 发送消息（CrimeFragment.java）

```
private Crime mCrime;
private EditText mTitleField;
private Button mDateButton;
private CheckBox mSolvedCheckbox;
private Button mReportButton;
...
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...

    mReportButton = (Button) v.findViewById(R.id.crime_report);
    mReportButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_SEND);
            i.setType("text/plain");
            i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
            i.putExtra(Intent.EXTRA_SUBJECT,
                    getString(R.string.crime_report_subject));
```

```
            startActivity(i);
        }
    });

    return v;
}
```

以上代码使用了一个接受字符串参数的`Intent`构造方法，我们传入的是一个定义操作的常量。取决于要创建的隐式intent类别，还有一些其他形式的构造方法可用。可以查阅`Intent`参考文档进一步了解。因为没有接受数据类型的构造方法可用，所以必须专门设置它。

消息内容和主题是作为extra附加到intent上的。注意，这些extra信息使用了`Intent`类中定义的常量。因此，任何响应该intent的activity都知道这些常量，自然也知道该如何使用它们的关联值。

运行CriminalIntent应用并点SEND CRIME REPORT按钮。因为刚创建的intent会匹配设备上的许多activity，所以你很可能会看到长长的候选activity列表，如图15-3所示。

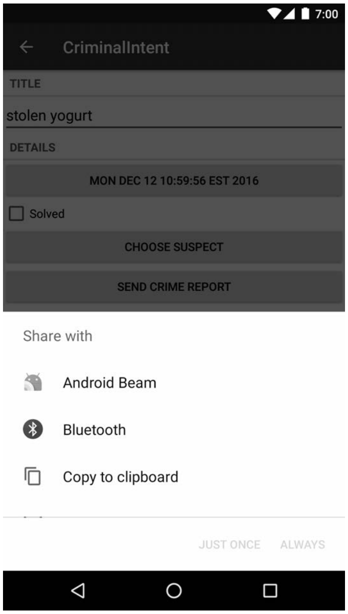


图15-3　支持发送消息的全部activity

从列表中作出选择后，可以看到消息加载到了所选应用中。接下来，只需填入地址，点击发送即可。

然而，有时可能看不到候选activity列表。出现这种情况通常有两个原因：要么是针对某个隐式intent设置了默认响应应用，要么是设备上仅有一个activity可以响应隐式intent。

通常，对于某项操作，最好使用用户的默认应用。不过，在CriminalIntent应用中，针对`ACTION_SEND`操作，应该总是将选择权交给用户。要知道，也许今天用户想低调处理问题，只采

取邮件的形式发送陋习报告，而明天很可能就改变主意了：更希望通过Twitter公开抨击那些公共场所的陋习。

使用隐式intent启动activity时，也可以创建每次都显示的activity选择器。和以前一样创建隐式intent后，调用以下`Intent`方法并传入创建的隐式intent以及用作选择器标题的字符串：

```
public static Intent createChooser(Intent target, String title)
```

然后，将`createChooser(...)`方法返回的intent传入`startActivity(...)`方法。

在CrimeFragment.java中，创建一个选择器显示响应隐式intent的全部activity，如代码清单15-10所示。

代码清单15-10　使用选择器（CrimeFragment.java）

```
public View onCreateView(layoutinflater inflater, ViewGroup container,
        Bundle savedinstanceState) {
    ...
    mReportButton.setOnClicklistener(new View.OnClicklistener() {
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_SEND);
            i.setType("text/plain");
            i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
            i.putExtra(Intent.EXTRA_SUBJECT,
                    getString(R.string.crime_report_subject));
        i = Intent.createChooser(i, getString(R.string.send_report));
        startActivity(i);
        }
```

运行CriminalIntent应用并点击SEND CRIME REPORT按钮。可以看到，只要有多个activity可以处理隐式intent，就会得到一个候选activity列表，如图15-4所示。

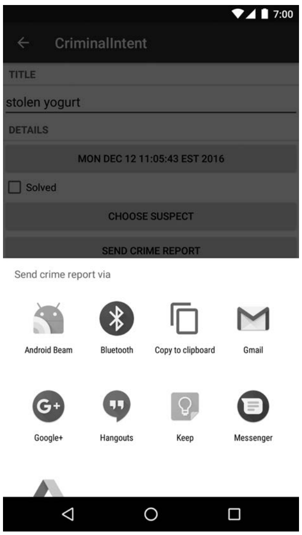


图15-4　通过选择器选择应用发送消息

15.4.3　获取联系人信息

现在，创建另一个隐式intent，让用户从联系人应用里选择嫌疑人。这个隐式intent将由操作以及数据获取位置组成。操作为`Intent.ACTION_PICK`。联系人数据获取位置为`ContactsContract.Contacts.CONTENT_URI`。简而言之，就是请Android帮忙从联系人数据库里获取某个具体的联系人。

因为要获取启动activity的返回结果，所以我们调用`startActivityForResult(...)`方法并传入intent和请求代码。在CrimeFragment.java中，新增请求代码常量和按钮成员变量，如代码清单15-11所示。

代码清单15-11　添加嫌疑人按钮成员变量（CrimeFragment.java）

```
private static final int REQUEST_DATE = 0;
private static final int REQUEST_CONTACT = 1;
...
private Button mSuspectButton;
private Button mReportButton;
```

在`onCreateView(...)`方法的末尾，引用新增按钮并为其设置监听器。在监听器接口实现中，创建一个隐式intent并传入`startActivityForResult(...)`方法。最后，如果找到联系人，就将其名字显示在按钮上，如代码清单15-12所示。

代码清单15-12　发送隐式intent（CrimeFragment.java）

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    final Intent pickContact = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI);
    mSuspectButton = (Button) v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            startActivityForResult(pickContact, REQUEST_CONTACT);
        }
    });

    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }

    return v;
}
```

稍后还会使用`pickContact`，所以这里没有将它放在`OnClickListener`监听器代码中。

运行CriminalIntent应用并点击CHOOSE SUSPECT按钮，应该能看到一个类似图15-5所示的联系人列表。

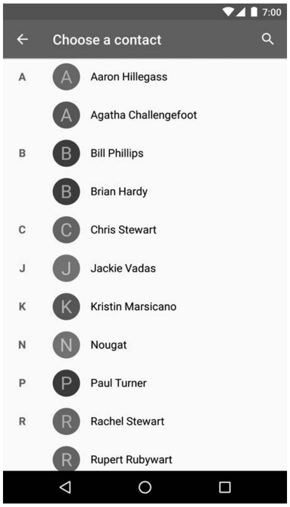


图15-5 包含嫌疑人的联系人列表

注意，如果设备上安装了其他联系人应用，应用界面可能会有所不同。另外可以看到，从当前应用中调用联系人应用时，完全不用知道应用的名字。因此，用户可以安装任何喜爱的联系人应用，操作系统会负责找到并启动它。

1. 从联系人列表中获取联系人数据

现在，需要从联系人应用中获取返回结果。很多应用都会共享联系人信息，因此Android提供了一个深度定制的API用于处理联系人信息：`ContentProvider`类。该类的实例封装了联系人数据库并提供给其他应用使用。我们可以通过`ContentResolver`访问`ContentProvider`。

前面，我们以`ACTION_PICK`启动了activity并要求返回结果，因此调用`onActivityResult(...)`方法会接收到一个intent。该intent包括了数据URI。这个URI是个数据定位符，指向用户所选的联系人。

在CrimeFragment.java中，将代码清单15-13所示的代码添加到`onActivityResult(...)`实现方法中。

代码清单15-13 获取联系人姓名（CrimeFragment.java）

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        return;
    }

    if (requestCode == REQUEST_DATE) {
        ...
        updateDate();

    } else if (requestCode == REQUEST_CONTACT && data != null) {
        Uri contactUri = data.getData();
```

15

```
            // Specify which fields you want your query to return
            // values for.
            String[] queryFields = new String[] {
                    ContactsContract.Contacts.DISPLAY_NAME
            };
            // Perform your query - the contactUri is like a "where"
            // clause here
            Cursor c = getActivity().getContentResolver()
                    .query(contactUri, queryFields, null, null, null);

            try {
                // Double-check that you actually got results
                if (c.getCount() == 0) {
                    return;
                }

                // Pull out the first column of the first row of data -
                // that is your suspect's name
                c.moveToFirst();
                String suspect = c.getString(0);
                mCrime.setSuspect(suspect);
                mSuspectButton.setText(suspect);
            } finally {
                c.close();
            }
        }
    }
```

以上代码创建了一条查询语句，要求返回全部联系人的名字。然后查询联系人数据库，获得一个可用的`Cursor`。因为已经知道`Cursor`只包含一条记录，所以将`Cursor`移动到第一条记录并获取它的字符串形式。该字符串即为嫌疑人的姓名。然后，使用它设置`Crime`嫌疑人，并显示在CHOOSE SUSPECT按钮上。

（联系人数据库是一个比较复杂的主题，这里不会展开讨论。如需详细了解，可以阅读Contacts Provider API指南：developer.android.com/guide/topics/providers/contacts-provider.html。）

现在可以运行应用测试了。有些设备可能没有联系人应用可用，如果是这样，请使用模拟器。

2. 联系人信息使用权限

如何获得读取联系人数据库的权限呢？实际上，这是联系人应用将其权限临时赋予了我们。联系人应用有使用联系人数据库的全部权限。联系人应用返回包含在intent中的URI数据给父activity时，会添加一个`Intent.FLAG_GRANT_READ_URI_PERMISSION`标志。该标志告诉Android，CriminalIntent应用中的父activity可以使用联系人数据一次。这很有用，因为不需要访问整个联系人数据库，我们只需要访问其中的一条联系人信息。

15.4.4　检查可响应任务的 activity

本章创建的第一个隐式intent总是会以某种方式得到响应，因为就算没有可用的消息发送应用，至少还会出现一个应用选择器；但第二个就不一定了，因为有些设备上根本就没有联系人应

用。如果操作系统找不到匹配的activity，应用就会崩溃。

解决办法是首先通过操作系统中的PackageManager类进行自检。在onCreateView(...)方法中实现检查，如代码清单15-14所示。

代码清单15-14　检查是否存在联系人应用（CrimeFragment.java）

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }

    PackageManager packageManager = getActivity().getPackageManager();
    if (packageManager.resolveActivity(pickContact,
            PackageManager.MATCH_DEFAULT_ONLY) == null) {
        mSuspectButton.setEnabled(false);
    }

    return v;
}
```

Android设备上安装了哪些组件以及包括哪些activity，PackageManager类全都知道。（本书后续章节还会介绍更多组件。）调用resolveActivity(Intent, int)方法，可以找到匹配给定Intent任务的activity。flag标志MATCH_DEFAULT_ONLY限定只搜索带CATEGORY_DEFAULT标志的activity。这和startActivity(Intent)方法类似。

如果搜到目标，它会返回ResolveInfo告诉我们找到了哪个activity。如果找不到的话，必须禁用嫌疑人按钮，否则应用就会崩溃。

如果想验证过滤器，但手头又没有不带联系人应用的设备，可临时添加额外的类别给intent。这个类别没有实际的作用，只是阻止任何联系人应用和你的intent匹配。过滤器验证代码如代码清单15-15所示。

代码清单15-15　过滤器验证代码（CrimeFragment.java）

```
public View onCreateView(layoutinflater inflater, ViewGroup container,
        Bundle savedinstanceState) {
    ...
    final Intent pickContact = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI);
    pickContact.addCategory(Intent.CATEGORY_HOME);
    mSuspectButton = (Button)v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        ...
    });
```

现在，嫌疑人按钮应该被禁用了，如图15-6所示。

15

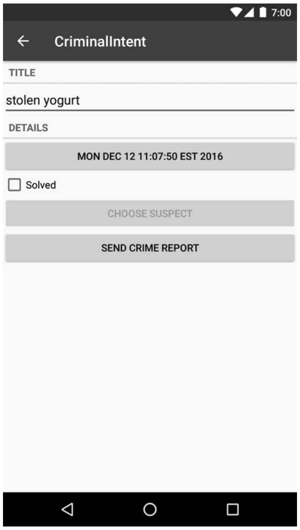


图15-6　嫌疑人选取按钮已禁用

验证完毕，记得删除相关代码，如代码清单15-16所示。

代码清单15-16　删除验证代码（CrimeFragment.java）

```
public View onCreateView(layoutinflater inflater, ViewGroup container,
        Bundle savedinstanceState) {
    ...
    final Intent pickContact = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI);
    pickContact.addCategory(Intent.CATEGORY_HOME);
    mSuspectButton = (Button)v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        ...
    });
```

15.5　挑战练习：ShareCompat

这个练习比较简单。Android支持库有个叫作ShareCompat的类，它有一个内部类叫IntentBuilder。使用这个内部类创建发送消息的Intent略微方便一些。

因此，请在mReportButton的监听器中，改用ShareCompat.IntentBuilder来创建你的Intent。

15.6　挑战练习：又一个隐式 intent

相较于发送消息，愤怒的用户可能更倾向于直接责问陋习嫌疑人。新增一个按钮，允许用户直接拨打陋习嫌疑人的电话。

要完成这个挑战，首先需要联系人数据库中的手机号码。这需要查询ContactsContract数据库中的CommonDataKinds.Phone表。如何查询，请查看它们的参考文档。

小提示：你应该使用android.permission.READ_CONTACTS权限。利用这个权限，可以查询到ContactsContract.Contacts._ID。然后再使用联系人ID查询CommonDataKinds.Phone表。

搞定了电话号码，就可以使用电话URI创建一个隐式intent：

```
Uri number = Uri.parse("tel:5551234");
```

与打电话相关的Intent操作有两种：Intent.ACTION_DIAL和Intent.ACTION_CALL。ACTION_CALL直接调出手机应用并拨打来自intent的电话号码；而ACTION_DIAL则拨好电话号码，然后等用户发起通话。

推荐使用ACTION_DIAL操作。这样的话，用户就有了冷静下来改变主意的机会。这种贴心的设计应该会受到欢迎的。

第 16 章 使用intent拍照

掌握了隐式intent之后，可考虑进一步丰富crime记录细节了。例如，给陋习现场拍张照片就是个不错的主意。

拍照需要用到一些包括隐式intent在内的新工具。隐式intent可以启动用户喜爱的相机应用并接收它拍摄的照片。

接收到照片后，该如何存储和展示这些照片呢？答案就在本章。

16.1 布置照片

首先要做的是在用户界面上布置照片。这需要新增两个`View`对象：显示照片缩略图的`ImageView`和拍照按钮。完成后的用户界面如图16-1所示。



图16-1 重新布置的用户界面

在同一行放置照片缩略图和拍照按钮的话，应用界面就会显得拥挤，给人不专业的感觉。下面就来合理地布置这两个视图。

参照图16-2所示的布局示意图，把新视图放入fragment_crime.xml布局。从左手边开始，首先添加ImageView视图用来显示照片，再添加ImageButton视图用来拍照。

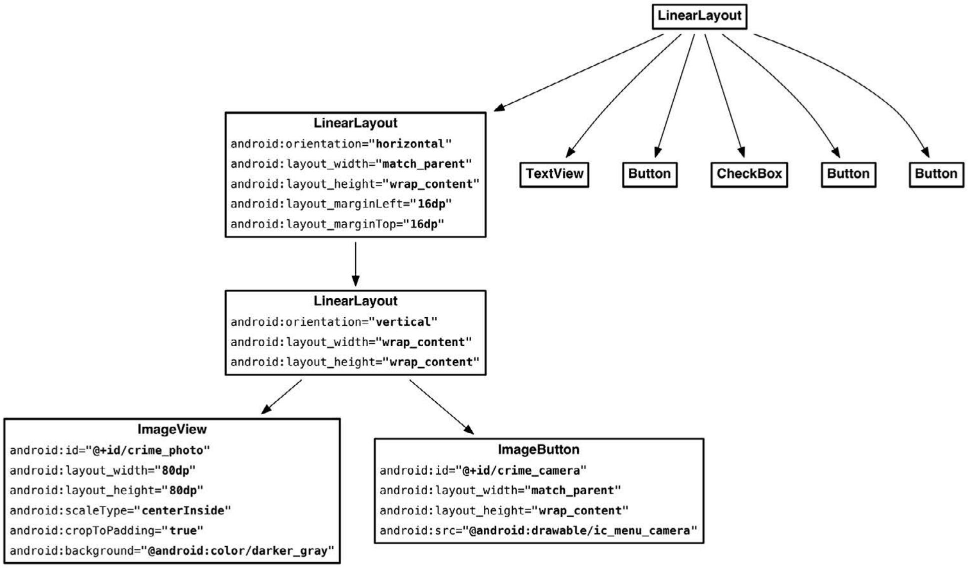


图16-2　相机视图布局（res/layout/fragment_crime.xml）

然后，参照图16-3所示的示意图，从右手边开始，把TextView标题栏和EditText文字框放入一个新LinearLayout布局中，再安排其作为图16-2新建的LinearLayout布局的子布局。

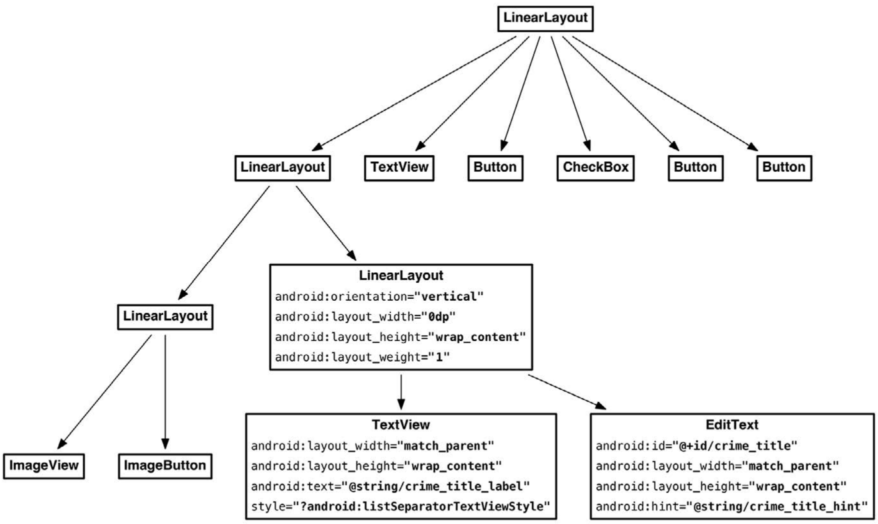


图16-3　标题栏布局（res/layout/fragment_crime.xml）

16

运行CriminalIntent应用，应该可以看到如图16-1所示的应用界面。

漂亮的用户界面完成了，但要响应ImageButton按钮点击和控制ImageView视图的内容展示，我们还要添加引用它们的实例变量。和以前一样，调用findViewById(int)方法从fragment_crime.xml布局中找到相应视图并使用它们，如代码清单16-1所示。

代码清单16-1　添加实例变量（CrimeFragment.java）

```
private Button mSuspctButton;
private Button mReportButton;
private ImageButton mPhotoButton;
private ImageView mPhotoView;
...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    PackageManager packageManager = getActivity().getPackageManager();
    if (packageManager.resolveActivity(pickContact,
            PackageManager.MATCH_DEFAULT_ONLY) == null) {
        mSuspectButton.setEnabled(false);
    }

    mPhotoButton = (ImageButton) v.findViewById(R.id.crime_camera);
    mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);

    return v;
}
```

与用户界面相关的工作完成了。接下来的任务是编码实现拍照和显示照片功能。

16.2　文件存储

相机所拍摄的照片动辄几MB大小，保存在SQLite数据库中肯定不现实。显然，它们需要在设备文件系统的某个地方保存。

很好，设备上就有这么一个地方：私有存储空间。还记得吗？前面，我们在私有存储空间保存过SQLite数据文件。使用类似Context.getFileStreamPath(String)和Context.getFilesDir()这样的方法，像照片这样的文件也可以这么保存。（结果就是照片文件保存在databases子目录相邻的某个子目录中。）

Context类提供的基本文件和目录处理方法如下。

- File getFilesDir()

  获取/data/data/<包名>/files目录。
- FileInputStream openFileInput(String name)

  打开现有文件进行读取。
- FileOutputStream openFileOutput(String name, int mode)

  打开文件进行写入，如果不存在就创建它。

- ❑ `File getDir(String name, int mode)`
  获取/data/data/<包名>/目录的子目录（如果不存在就先创建它）。
- ❑ `String[] fileList()`
  获取主文件目录下的文件列表。可与其他方法配合使用，如`openFileInput(String)`。
- ❑ `File getCacheDir()`
  获取/data/data/<包名>/cache目录。应注意及时清理该目录，并节约使用。

如果存储的文件仅供应用内部使用，使用上述各类方法就够了。

如果其他应用要读写你的文件，事情就没那么简单了。CriminalIntent应用就是这个情况：外部相机应用需要在你的应用里保存拍摄的照片。虽然有个`Context.MODE_WORLD_READABLE`可以传入`openFileOutput(String, int)`方法，但这个flag已经废弃了。即使强制使用，在新系统设备上也不是那么可靠。以前，还有个办法是通过公共外部存储转存，但出于安全考虑，这条路在新版本系统上也被堵住了。

如果想共享文件给其他应用，或是接收其他应用的文件（如相机应用拍摄的照片），可以通过`ContentProvider`把要共享的文件暴露出来。`ContentProvider`允许你暴露内容URI给其他应用。这样，这些应用就可以从内容URI下载或向其中写入文件。当然，主动权在你手上，你可以控制读或写。

16.2.1　使用 FileProvider

如果只想从其他应用接收一个文件，自己实现`ContentProvider`简直是费力不讨好的事。Google早已想到这点，所以提供了一个名叫`FileProvider`的便利类。这个类能帮你搞定一切，而你只要做做参数配置就行了。

首先，声明`FileProvider`为`ContentProvider`，并给予一个指定的权限。在AndroidManifest.xml中添加一个`FileProvider`声明，如代码清单16-2所示。

代码清单16-2　添加FileProvider声明（AndroidManifest.xml）

```
<activity
    android:name=".CrimePagerActivity"
    android:parentActivityName=".CrimelistActivity">
</activity>
<provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="com.bignerdranch.android.criminalintent.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
</provider>
```


这里的权限是指一个位置：文件保存地。把`FileProvider`和你指定的位置关联起来，就相当于你给发出请求的其他应用一个目标地。添加`exported = "false"`属性就意味着，除了你自己以及你给予授权的人，其他任何人都不允许使用你的`FileProvider`。而`grantUriPermissions`属性用来给其他应用授权，允许它们向你指定位置的URI（稍后你会看到，这个位置信息放在intent

中对外发出）写入文件。

既然已让Android知道FileProvider在哪，还需要配置FileProvider，让它知道该暴露哪些文件。这个配置用另外一个XML资源文件处理。在项目工具窗口，右键点击app/res目录，然后选择New → Android resource file菜单项。资源类型选XML，文件名输入files，确认并创建这个文件。

打开刚刚创建的xml/files.xml文件，切换到代码模式，按代码清单16-3替换原有内容。

代码清单16-3　填写路径描述（res/xml/files.xml）

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

</PreferenceScreen>
<paths>
    <files-path name="crime_photos" path="."/>
</paths>
```

这是个描述性XML文件，其表达的意思是，把私有存储空间的根路径映射为crime_photos。这个名字仅供FileProvider内部使用，你不应去用它。

最后，在AndroidManifest.xml文件中，添加一个meta-data标签，让FileProvider能找到files.xml文件，如代码清单16-4所示。

代码清单16-4　关联使用路径描述资源（AndroidManifest.xml）

```
<provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="com.bignerdranch.android.criminalintent.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/files"/>
</provider>
```

16.2.2　指定照片存放位置

现在要处理的是指定照片存放位置。首先，在Crime.java中添加获取文件名的方法，如代码清单16-5所示。

代码清单16-5　添加文件名获取方法（Crime.java）

```
    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }

    public String getPhotoFilename() {
        return "IMG_" + getId().toString() + ".jpg";
    }
}
```

Crime.getPhotoFilename()方法不知道照片文件该存储在哪个目录。不过，既然文件名基

于Crime ID编制，它具有唯一性。

接下来，找到要保存文件的目录。CrimeLab负责CriminalIntent应用的数据持久化工作。既然如此，那么在CrimeLab类里添加getPhotoFile(Crime)方法也就再合适不过了，如代码清单16-6所示。

代码清单16-6 定位照片文件（CrimeLab.java）

```
public class Crimelab {
    ...
    public Crime getCrime(UUiD id) {
        ...
    }

    public File getPhotoFile(Crime crime) {
        File filesDir = mContext.getFilesDir();
        return new File(filesDir, crime.getPhotoFilename());
    }

    public void updateCrime(Crime crime) {
        ...
    }
```

上述新增方法不会创建任何文件。它的作用就是返回指向某个具体位置的File对象。稍后，我们会使用FileProvider把路径以URI的形式对外暴露。

16.3 使用相机 intent

现在可以实现拍照功能了。这并不难，只要使用一个隐式intent就可以了。

首先是保存照片文件存储位置，如代码清单16-7所示。（接下来好几个地方会用到它，做好这步会省不少事。）

代码清单16-7 获取照片文件位置（CrimeFragment.java）

```
private Crime mCrime;
private File mPhotoFile;
private EditText mTitleField;
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    mPhotoFile = CrimeLab.get(getActivity()).getPhotoFile(mCrime);
}
```

16

然后是处理相机拍照按钮，触发拍照。相机intent定义在MediaStore里。这个类负责处理所有与多媒体相关的任务。发送一个带MediaStore.ACTION_IMAGE_CAPTURE操作的intent，Android会启动相机activity拍照。

实现拍照功能的思路已经理清了，但还有些小细节要处理。

触发拍照

准备工作都已完成，可以使用相机intent了。我们需要的intent操作是定义在MediaStore类中的ACTION_IMAGE_CAPTURE。MediaStore类定义了一些公共接口，可用于处理图像、视频以及音乐这些常见的多媒体任务。当然，这也包括触发相机应用的拍照intent。

ACTION_IMAGE_CAPTURE打开相机应用，默认只能拍摄缩略图这样的低分辨率照片，而且照片会保存在onActivityResult(...)返回的Intent对象里。

要想获得全尺寸照片，就要让它使用文件系统存储照片。这可以通过传入保存在MediaStore.EXTRA_OUTPUT中的指向存储路径的Uri来完成。这个Uri会指向FileProvider提供的位置。

编写用于拍照的隐式intent，如代码清单16-8所示。拍摄的照片应该保存在mPhotoFile指定的地方。同时，别忘了检查设备上是否安装有相机应用，以及是否有地方存储照片。（要确认是否有可用的相机应用，可找PackageManager确认是否有响应相机隐式intent的activity。关于查询PackageManager的详细内容，参见15.4.4节。）

代码清单16-8　使用相机intent（CrimeFragment.java）

```
private static final int REQUEST_DATE = 0;
private static final int REQUEST_CONTACT = 1;
private static final int REQUEST_PHOTO= 2;
...
@Override
public View onCreateView(layoutinflater inflater, ViewGroup container,
        Bundle savedinstanceState) {
    ...
    mPhotoButton = (imageButton) v.findViewByid(R.id.crime_camera);
    final Intent captureImage = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    boolean canTakePhoto = mPhotoFile != null &&
            captureImage.resolveActivity(packageManager) != null;
    mPhotoButton.setEnabled(canTakePhoto);

    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Uri uri = FileProvider.getUriForFile(getActivity(),
                    "com.bignerdranch.android.criminalintent.fileprovider", mPhotoFile);
            captureImage.putExtra(MediaStore.EXTRA_OUTPUT, uri);

            List<ResolveInfo> cameraActivities = getActivity()
                    .getPackageManager().queryIntentActivities(captureImage,
                            PackageManager.MATCH_DEFAULT_ONLY);

            for (ResolveInfo activity : cameraActivities) {
                getActivity().grantUriPermission(activity.activityInfo.packageName,
                        uri, Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
            }
```

```
            startActivityForResult(captureImage, REQUEST_PHOTO);

        }
    });

    mPhotoView = (imageView) v.findViewByid(R.id.crime_photo);

    return v;
}
```

调用FileProvider.getUriForFile(...)会把本地文件路径转换为相机能看见的Uri形式。要实际写入文件，还需要给相机应用权限。为了授权，我们授予FLAG_GRANT_WRITE_URI_PERMISSION给所有cameraImage intent的目标activity，以此允许它们在Uri指定的位置写文件。当然，还有个前提条件：在声明FileProvider的时候添加过android:grantUriPermissions属性。

运行CriminalIntent应用，点击相机按钮启动相机应用，如图16-4所示。



图16-4　打开设备上的相机应用

16.4　缩放和显示位图

现在，终于可以拍摄陋习现场的照片并保存了。

有了照片，接下来就是找到并加载它，然后展示给用户看。在技术实现上，这需要加载照片到大小合适的Bitmap对象中。要从文件生成Bitmap对象，我们需要BitmapFactory类：

```
Bitmap bitmap = BitmapFactory.decodeFile(mPhotoFile.getPath());
```

看到这里，有没有感觉不对劲？肯定有的。否则依照本书代码风格，上述代码就会直接加粗印刷，你对照输入就行了。

不卖关子了，问题在于，介绍Bitmap时，我们提到"大小合适"。Bitmap是个简单对象，它只存储实际像素数据。也就是说，即使原始照片已压缩过，但存入Bitmap对象时，文件并不会同样压缩。因此，一张1600万像素24位的相机照片（存为JPG格式大约5MB），一旦载入Bitmap对象，就会立即膨胀至48MB！

这个问题可以设法解决，但需要手动缩放位图照片。具体做法就是，首先确认文件到底有多大，然后考虑按照给定区域大小合理缩放文件。最后，重新读取缩放后的文件，创建Bitmap对象。

创建名为PictureUtils的新类，并在其中添加getScaledBitmap(String, int, int)缩放方法，如代码清单16-9所示。

代码清单16-9　创建getScaledBitmap(...)方法（PictureUtils.java）

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, int destWidth, int destHeight) {
        // Read in the dimensions of the image on disk
        BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFile(path, options);

        float srcWidth = options.outWidth;
        float srcHeight = options.outHeight;

        // Figure out how much to scale down by
        int inSampleSize = 1;
        if (srcHeight > destHeight || srcWidth > destWidth) {
            float heightScale = srcHeight / destHeight;
            float widthScale = srcWidth / destWidth;

            inSampleSize = Math.round(heightScale > widthScale ? heightScale :
                    widthScale);
        }

        options = new BitmapFactory.Options();
        options.inSampleSize = inSampleSize;

        // Read in and create final bitmap
        return BitmapFactory.decodeFile(path, options);
    }
}
```

上述方法中，inSampleSize值很关键。它决定着缩略图像素的大小。假设这个值是1的话，就表明缩略图和原始照片的水平像素大小一样。如果是2的话，它们的水平像素比就是1∶2。因此，inSampleSize值为2时，缩略图的像素数就是原始文件的四分之一。

问题总是接踵而来。解决了缩放问题，又冒出了新问题：fragment刚启动时，无人知道PhotoView究竟有多大。onCreate(...)、onStart()和onResume()方法启动后，才会有首个

实例化布局出现。也就在此时，显示在屏幕上的视图才会有大小尺寸。这也是出现新问题的原因。

解决方案有两个：要么等布局实例化完成并显示，要么干脆使用保守估算值。特定条件下，尽管估算比较主观，但确实是唯一切实可行的办法。再添加一个getScaledBitmap(String, Activity)静态Bitmap估算方法，如代码清单16-10所示。

代码清单16-10　编写合理的缩放方法（PictureUtils.java）

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, Activity activity) {
        Point size = new Point();
        activity.getWindowManager().getDefaultDisplay()
                .getSize(size);

        return getScaledBitmap(path, size.x, size.y);
    }
```

该方法先确认屏幕的尺寸，然后按此缩放图像。这样，就能保证载入的ImageView永远不会过大。无论如何，这是一个比较保守的估算，但能解决问题。

接下来，为了把Bitmap载入ImageView，在CrimeFragment.java中，添加刷新mPhotoView的方法，如代码清单16-11所示。

代码清单16-11　更新mPhotoView（CrimeFragment.java）

```
    private String getCrimeReport() {
        ...
    }

    private void updatePhotoView() {
        if (mPhotoFile == null || !mPhotoFile.exists()) {
            mPhotoView.setImageDrawable(null);
        } else {
            Bitmap bitmap = PictureUtils.getScaledBitmap(
                    mPhotoFile.getPath(), getActivity());
            mPhotoView.setImageBitmap(bitmap);
        }
    }
}
```

然后，分别在onCreateView(...)方法和onActivityResult(...)方法中调用updatePhotoView()方法，如代码清单16-12所示。

代码清单16-12　调用updatePhotoView()方法（CrimeFragment.java）

```
        mPhotoButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ...
                startActivityForResult(captureImage, REQUEST_PHOTO);
            }
        });
```

```
        mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);
        updatePhotoView();

        return v;
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }

        if (requestCode == REQUEST_DATE) {
            ...
        } else if (requestCode == REQUEST_CONTACT && data != null) {
            ...
        } else if (requestCode == REQUEST_PHOTO) {
            Uri uri = FileProvider.getUriForFile(getActivity(),
                    "com.bignerdranch.android.criminalintent.fileprovider",
                    mPhotoFile);

            getActivity().revokeUriPermission(uri,
                    Intent.FLAG_GRANT_WRITE_URI_PERMISSION);

            updatePhotoView();
        }
    }
```

既然相机已保存了文件，那就再次调用权限，关闭文件访问。再次运行应用，应该可以看到已拍照片的缩略图了。

16.5　功能声明

应用的拍照功能用起来不错，但还有一件事情要做：告诉潜在用户应用有拍照功能。

假如应用要用到诸如相机、NFC，或者任何其他的随设备走的功能时，都应该要让Android系统知道。这样，假如设备缺少这样的功能，类似Google Play商店的安装程序就会拒绝安装应用。

为了声明应用要使用相机，在AndroidManifest.xml中加入<uses-feature>标签，如代码清单16-13所示。

代码清单16-13　添加<uses-feature>标签（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.criminalintent" >

    <uses-feature android:name="android.hardware.camera"
                  android:required="false"
        />
```

注意，我们在代码中使用了android:required属性。默认情况下，声明要使用某个设备功能后，应用就无法支持那些无此功能的设备了，但这不适用于CriminalIntent应用。这是因为，

resolveActivity(...)方法可以判断设备是否支持拍照。如果不支持，就直接禁用拍照按钮。

无论如何，这里设置android:required属性为false，Android系统因此就知道，尽管不带相机的设备会导致应用功能缺失，但应用仍然可以正常安装和使用。

16.6 挑战练习：优化照片显示

现在虽然能够看到拍摄的照片，但没法看到照片细节。

请创建能显示放大版照片的DialogFragment。只要点击缩略图，就会弹出这个DialogFragment，让用户查看放大版的照片。

16.7 挑战练习：优化缩略图加载

本章，我们只能大致估算缩略图的目标尺寸。虽说这种做法可行且实施迅速，但还不够理想。

Android有个现成的API工具可用，叫作ViewTreeObserver。你可以从Activity层级结构中获取任何视图的ViewTreeObserver对象：

```
ViewTreeObserver observer = mImageView.getViewTreeObserver();
```

你可以为ViewTreeObserver对象设置包括OnGlobalLayoutListener在内的各种监听器。使用OnGlobalLayoutListener监听器，可以监听任何布局的传递，控制事件的发生。

调整代码，使用有效的mPhotoView尺寸，等到有布局切换时再调用updatePhotoView()方法。

第 17 章

双版面主从用户界面

本章，为了适应平板设备，我们改造CriminalIntent应用的用户界面，让用户能同时看到列表和明细界面并与它们交互。图17-1展示了这样的列表明细界面，也可称其为*主从用户界面*（master-detail interface）。

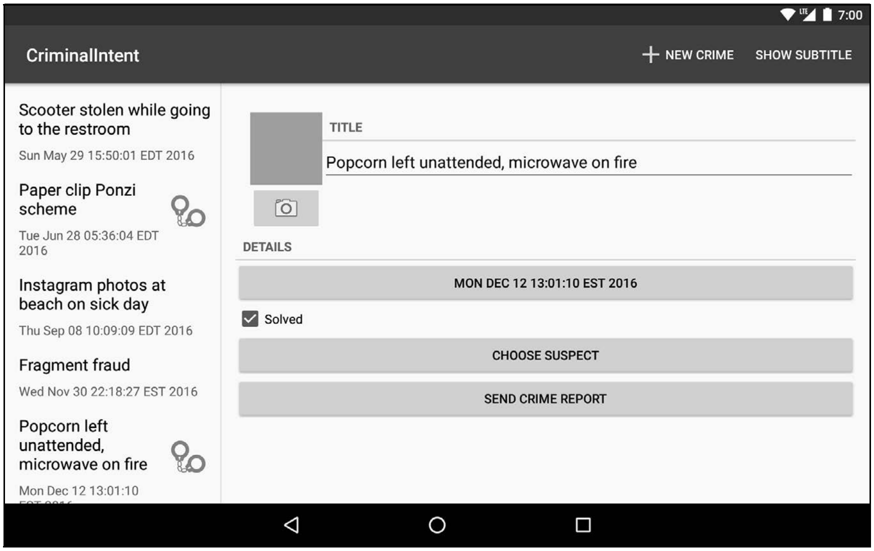


图17-1　同时显示列表和明细的用户界面

本章的代码验证需要使用平板设备或AVD。要创建平板AVD，首先选择Tools → Android → AVD Manager菜单项，然后点击+Create Virtual Device...按钮，在弹出的界面选择Tablet类别。选择目标虚拟硬件配置后，点击Next按钮继续，如图17-2所示。最后，确认API级别至少为21，点击Finish按钮完成。

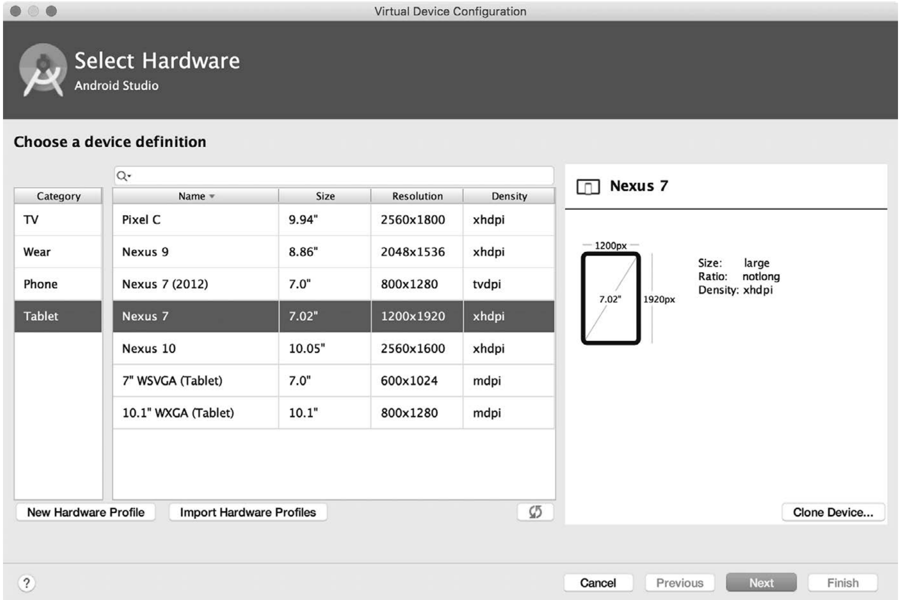


图17-2　AVD平板设备选择

17.1　增加布局灵活性

在手机设备上，CrimeListActivity生成的是单版面（single-pane）布局。在平板设备上，为了同时显示主从视图，我们需要它生成双版面（two-pane）布局。

在双版面布局中，CrimeListActivity将同时托管CrimeListFragment和CrimeFragment，如图17-3所示。

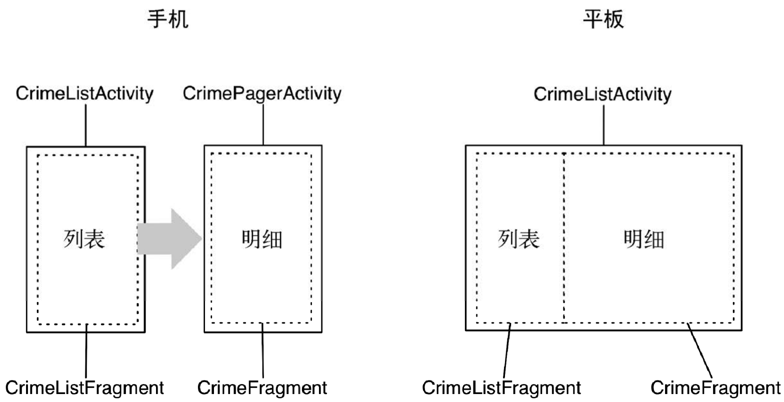


图17-3　不同类型的布局

要实现双版面布局，需完成如下任务：

- 修改SingleFragmentActivity，不再硬编码实例化布局；
- 创建包含两个fragment容器的布局；
- 修改CrimeListActivity，实现在手机设备上实例化单版面布局，在平板设备上实例化双版面布局。

17.1.1　修改 SingleFragmentActivity

CrimeListActivity是SingleFragmentActivity的子类。当前，SingleFragmentActivity只能实例化activity_fragment.xml布局。为了使SingleFragmentActivity类更加灵活易用，我们让它的子类自己提供布局资源ID。

在SingleFragmentActivity.java中，添加一个protected方法，返回activity需要的布局资源ID，如代码清单17-1所示。

代码清单17-1　增加SingleFragmentActivity类的灵活性（SingleFragmentActivity.java）

```
public abstract class SingleFragmentActivity extends AppCompatActivity {

    protected abstract Fragment createFragment();

    @LayoutRes
    protected int getLayoutResId() {
        return R.layout.activity_fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        setContentView(getLayoutResId());

        FragmentManager fm = getSupportFragmentManager();
        ...
    }
}
```

现在，虽然SingleFragmentActivity抽象类的功能和以前一样，但是，如果不想再使用固定不变的activity_fragment.xml布局，它的子类可以选择覆盖getLayoutResId()方法返回所需布局。getLayoutResId()方法使用了@LayoutRes注解。这告诉Android Studio，任何时候，该实现方法都应该返回有效的布局资源ID。

17.1.2　创建包含两个 fragment 容器的布局

在项目工具窗口中，右键单击res/layout/目录，新建一个XML文件。在弹出的新建XML文件界面，指定资源类型为Layout，命名文件为activity_twopane.xml，然后选择LinearLayout作为根元素，最后单击Finish按钮完成。

参照图17-4，完成双版面布局的XML内容定义。

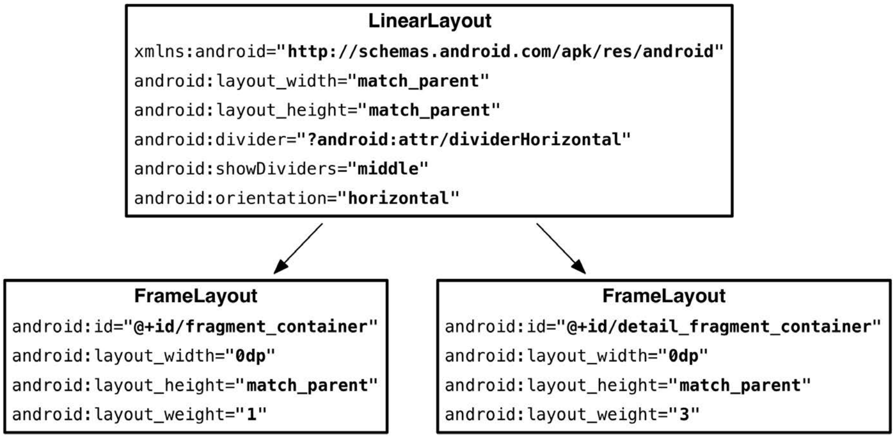


图17-4 包含两个fragment容器的布局（layout/activity_twopane.xml）

注意，布局定义的第一个`FrameLayout`也有一个`fragment_container`布局资源ID，因此`SingleFragmentActivity.onCreate(...)`方法的相关代码能够像以前一样工作。activity创建后，`createFragment()`方法返回的fragment将会出现在屏幕左侧的版面中。

要测试新建布局，在`CrimeListActivity`类中覆盖`getLayoutResId()`方法，返回`R.layout.activity_twopane`资源ID，如代码清单17-2所示。

代码清单17-2 使用双版面布局（CrimeListActivity.java）

```
public class CrimeListActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
    protected int getLayoutResId() {
        return R.layout.activity_twopane;
    }
}
```

在平板设备或AVD上运行CriminalIntent应用，确认可以看到如图17-5所示的用户界面。注意，右边的明细版面什么也没显示。即使点击任意列表项，也无法显示对应的crime明细信息。本章稍后会完成crime明细fragment容器的编码及设置工作。

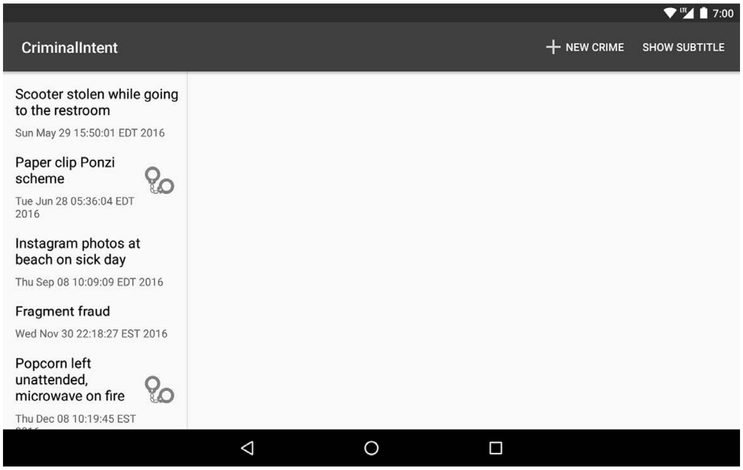


图17-5 平板设备上的双版面布局

当前，无论是在手机还是在平板设备上，`CrimeListActivity`都会生成双版面的用户界面。下一节将使用别名资源来解决这个问题。

17.1.3 使用别名资源

*别名资源*是一种指向其他资源的特殊资源。它存放在res/values/目录下，并按照约定定义在refs.xml文件中。

接下来的任务就是让`CrimeListActivity`基于不同的设备使用不同的布局文件。这实际类似于前面章节对水平布局和竖直布局的选择和控制：使用资源修饰符。

让res/layout/目录中的文件使用资源修饰符虽然可行，但也有缺点。最明显的缺点就是数据冗余，因为每个布局文件都要复制一份。例如，如果想使用activity_masterdetail.xml布局文件，就需要将activity_fragment.xml复制到res/layout/activity_masterdetail.xml中，并且将activity_twopane.xml复制到res/layout-sw600dp/activity_masterdetail.xml中（稍后会提到sw600dp的作用）。

要解决上述问题，可以使用别名资源。本小节将分别创建用于手机指向activity_fragment. xml布局的别名资源，以及用于平板指向activity_twopane.xml布局的别名资源。

在项目工具窗口中，右键单击res/values/目录，新建values资源文件。在弹出的新建XML文件界面，选择资源类型为`Values`，并将文件命名为refs.xml。确认新建的文件不带任何修饰符，最后单击Finish按钮。参照代码清单17-3，在新建的refs.xml中添加item节点定义。

代码清单17-3 创建默认的别名资源值（res/values/refs.xml）

```
<resources>

  <item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>

</resources>
```

别名资源指向了单版面布局资源文件。别名资源自身也具有资源ID：R.layout.activity_masterdetail。注意，别名的type属性决定资源ID属于什么内部类。即使别名资源自身在res/values/目录中，它的资源ID依然属于R.layout内部类。

修改CrimeListActivity类，以R.layout.activity_masterdetail资源ID替换R.layout.activity_fragment，如代码清单17-4所示。

代码清单17-4　再次切换布局（CrimeListActivity.java）

```
@Override
protected int getLayoutResId() {
    return R.layout.activity_twopane;masterdetail;
}
```

运行CriminalIntent应用并验证别名资源的使用。一切正常的话，CrimeListActivity应该再次生成了单版面布局。

17.1.4　创建平板设备专用可选资源

因为res/values/目录中的别名资源是系统默认的别名资源，所以CrimeListActivity生成了单版面布局。

现在，创建一个大屏幕设备专用的可选别名资源，让activity_masterdetail别名资源指向activity_twopane.xml双版面布局资源。

在项目工具窗口中，右键单击res/values/目录，弹出如图17-6所示的新建资源文件窗口。资源文件名和目录名依然分别是refs.xml和values，但这次要用>>按钮把Available qualifiers窗口中的Screen Width选到右边窗口去。

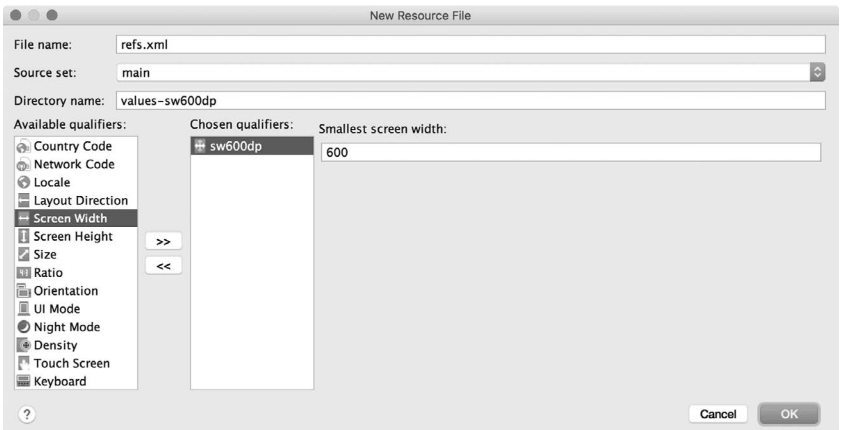


图17-6　添加资源修饰符

可以看到，资源修饰符处要求指定Smallest Screen Width的数值。输入600后点击OK按钮完成。在随后打开的新建资源文件中，添加activity_masterdetail别名资源指向activity_twopane.

xml，如代码清单17-5所示。

代码清单17-5　用于大屏幕设备的可选资源（res/values-sw600dp/refs.xml）

```
<resources>

  <item name="activity_masterdetail" type="layout">@layout/activity_twopane</item>

</resources>
```

对于上述新增别名资源，我们的目标是：

- 对于小于指定尺寸的设备，使用activity_fragment.xml资源文件；
- 对于大于指定尺寸的设备，使用activity_twopane.xml资源文件。

Android只提供一部分的资源适配机制。配置修饰符-sw600dp的作用是：如果设备尺寸大于某个指定值，就使用对应的资源文件。sw是smallest width（最小宽度）的缩写。虽然字面上是宽度的含义，但它实际指的是屏幕的最小尺寸（dimension），因而sw与设备的当前方向无关。

在确定可选资源时，-sw600dp配置修饰符表明：对任何最小尺寸为600dp或更高dp的设备，都使用该资源。对于指定平板的屏幕尺寸规格来说，这是一种非常好的做法。

那另一部分的资源适配怎么处理呢？对于希望使用activity_fragment.xml的小尺寸设备要怎么做呢？这好办，Android是这样判断的：既然设备尺寸小于-sw600dp配置修饰符的指定值，那就使用默认的activity_fragment.xml资源文件。

分别在手机和平板上运行CriminalIntent应用，确认单双版面布局都能正常显示。

17.2　activity：fragment 的托管者

处理完单双版面布局的显示，就可以着手将CrimeFragment添加给crime明细fragment容器，让CrimeListActivity展示一个完整的双版面用户界面。

你可能会认为，只需再为平板设备实现一个CrimeHolder.onClick(View)监听器方法就行了。这样，无需启动新的CrimePagerActivity，onClick(View)方法会获取CrimeListActivity的FragmentManager，然后提交一个fragment事务，将CrimeFragment添加到明细fragment容器中。

这种设想的具体实现代码如下（CrimeListFragment.CrimeHolder）：

```
public void onClick(View view) {
    // Stick a new CrimeFragment in the activity's layout
    Fragment fragment = CrimeFragment.newInstance(mCrime.getId());
    FragmentManager fm = getActivity().getSupportFragmentManager();
    fm.beginTransaction()
        .add(R.id.detail_fragment_container, fragment)
        .commit();
}
```

虽然行得通，但做法很老套。fragment天生是个独立的开发构件。如果要开发fragment用来添加其他fragment到activity的FragmentManager，那么这个fragment就必须知道托管activity是如何工作的。结果，该fragment就再也无法作为独立的开发构件使用了。

以上述代码为例，CrimeListFragment将CrimeFragment添加给了CrimeListActivity，并且认为CrimeListActivity的布局里包含detail_fragment_container。但实际上，CrimeListFragment根本就不应关心这些，这都是其托管activity应该处理的事情。

为了让fragment独立，我们可以在fragment中定义回调接口，委托托管activity来完成那些不应由fragment处理的任务。托管activity将实现回调接口，履行托管fragment的任务。

fragment 回调接口

要委托工作任务给托管activity，通常的做法是由fragment定义名为Callbacks的回调接口。回调接口定义了fragment委托给托管activity处理的工作任务。任何打算托管目标fragment的activity都必须实现它。

有了回调接口，就不用关心谁是托管者，fragment可以直接调用托管activity的方法。

1. 实现CrimeListFragment.Callbacks回调接口

为了实现Callbacks接口，首先要定义一个成员变量，用于存放实现Callbacks接口的对象。然后将托管activity强制类型转换为Callbacks对象并赋值给Callbacks类型变量。

activity赋值是在Fragment的生命周期方法中处理的：

```
public void onAttach(Context context)
```

该方法是在fragment附加给activity时调用的，当然fragment是否保留并不重要。记住，Activity是Context的子类，所以，onAttach可以传入Context参数，这确实很灵活。请确保使用onAttach(Context)方法，而不是已废弃的onAttach(Activity)方法（将来可能会被移除）。

类似地，在相应的生命周期销毁方法中，将Callbacks变量设置为null。

```
public void onDetach()
```

这里将变量清空的原因是，随后再也无法访问该activity或指望它继续存在了。

在CrimeListFragment.java中，添加Callbacks接口。另外添加一个mCallbacks变量并覆盖onAttach(Context)和onDetach()方法，完成变量的赋值与清空，如代码清单17-6所示。

代码清单17-6 添加回调接口（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {
    ...
    private boolean mSubtitleVisible;
    private Callbacks mCallbacks;

    /**
     * Required interface for hosting activities
     */
    public interface Callbacks {
        void onCrimeSelected(Crime crime);
    }

    @Override
    public void onAttach(Activity activity) {
```

```
        super.onAttach(Context);
        mCallbacks = (Callbacks) activity;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }
    ...
    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mCallbacks = null;
    }
}
```

现在，CrimeListFragment有办法调用托管activity方法了。另外，它也不关心托管activity是谁。只要托管activity实现了CrimeListFragment.Callbacks接口，CrimeListFragment中的一切代码行为就都可以保持不变。

注意，未经类安全性检查，CrimeListFragment就将托管activity强制转换为了CrimeList-Fragment.Callbacks对象。这意味着，托管activity必须实现CrimeListFragment.Callbacks接口。这并非是不良的依赖关系，但记录下它非常重要。

接下来，在CrimeListActivity类中，实现CrimeListFragment.Callbacks接口，如代码清单17-7所示。暂时不用理会onCrimeSelected(Crime)空方法，稍后再来处理。

代码清单17-7　实现回调接口（CrimeListActivity.java）

```
public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
    protected int getLayoutResId() {
        return R.layout.activity_masterdetail;
    }

    @Override
    public void onCrimeSelected(Crime crime) {
    }
}
```

最终，在用户创建新crime记录时，CrimeListFragment将在CrimeHolder.onClick(...)方法里调用onCrimeSelected(Crime)方法。现在，先思考如何实现CrimeListActivity.onCrimeSelected(Crime)方法。

onCrimeSelected(Crime)方法被调用时，CrimeListActivity需要完成以下二选一任务：

- 如果使用手机用户界面布局，启动新的CrimePagerActivity；
- 如果使用平板设备用户界面布局，将CrimeFragment放入detail_fragment_container中。

是实例化手机界面布局还是平板界面布局，可以检查布局ID；但是推荐检查布局是否包含detail_ fragment_container。这是因为布局文件名随时会变，并且我们也不关心布局是从哪个文件实例化产生。我们只需知道，布局文件是否包含可以放入CrimeFragment的detail_fragment_container。

如果包含，那就创建一个fragment事务，将我们需要的CrimeFragment添加到detail_fragment_container中。如果之前就有CrimeFragment，应首先移除它。

在CrimeListActivity.java中，实现onCrimeSelected(Crime)方法，按照不同的布局界面分别处理，如代码清单17-8所示。

代码清单17-8　有条件的CrimeFragment启动（CrimeListActivity.java）

```
@Override
public void onCrimeSelected(Crime crime) {
    if (findViewById(R.id.detail_fragment_container) == null) {
        Intent intent = CrimePagerActivity.newIntent(this, crime.getId());
        startActivity(intent);
    } else {
        Fragment newDetail = CrimeFragment.newInstance(crime.getId());

        getSupportFragmentManager().beginTransaction()
                .replace(R.id.detail_fragment_container, newDetail)
                .commit();
    }
}
```

最后，在CrimeListFragment类中，在启动新的CrimePagerActivity处调用onCrimeSelected(Crime)方法。

在CrimeListFragment.java中，修改CrimeHolder.onClick(View)和onOptionsItemSelected(MenuItem)，以调用Callbacks.onCrimeSelected(Crime)方法，如代码清单17-9所示。

代码清单17-9　调用全部回调方法（CrimeListFragment.java）

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent intent = CrimePagerActivity
                    .newIntent(getActivity(), crime.getId());
```

```
                startActivity(intent);
                updateUI();
                mCallbacks.onCrimeSelected(crime);
                return true;
            ...
        }
    }
    ...
    private class CrimeHolder extends RecyclerView.ViewHolder
            implements View.OnClickListener {
        ...
        @Override
        public void onClick(View view) {
            Intent intent = CrimePagerActivity.newIntent(getActivity(), mCrime.getId());
            startActivity(intent);
            mCallbacks.onCrimeSelected(mCrime);
        }
    }
```

在onOptionsItemSelected(MenuItem)方法中调用回调方法时，只要新增crime记录，crime列表就会立即重新加载。这很有必要，因为在平板设备上，新增crime记录后，crime列表依然可见；而在手机设备上，crime明细界面会在列表界面之前出现，列表项的刷新可以很灵活地处理。

在平板设备上运行CriminalIntent应用。添加一条crime记录，可以看到，detail_fragment_container容器中立即显示了新的CrimeFragment视图。然后，尝试查看其他旧记录以观察CrimeFragment视图的切换，如图17-7所示。

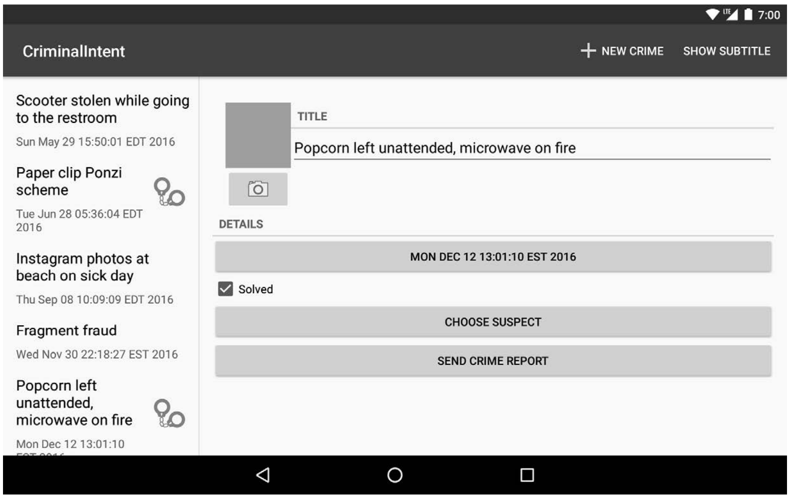


图17-7　已关联的主界面和从界面

看上去真不错！不过，还不够完美：如果修改crime明细内容，列表项并不会显示最新内容。当前，在CrimeListFragment.onResume()方法中，只有新添加crime记录，我们才能立即刷新显示列表项界面。但是在平板设备上，CrimeListFragment和CrimeFragment同时可见。因此，

当CrimeFragment出现时，CrimeListFragment不会暂停——没有暂停怎会有恢复？这就是crime列表项不能刷新的根本原因。

下面，我们在CrimeFragment中添加另一个回调接口来修正这个问题。

2. 实现CrimeFragment.Callbacks回调接口

CrimeFragment类中定义的接口如下：

```
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}
```

CrimeFragment如果要刷新数据，需要做两件事。首先，既然CriminalIntent应用的数据源是SQLite数据库，那么它需要把Crime保存到CrimeLab里。然后，CrimeFragment类会调用托管activity的onCrimeUpdated(Crime)方法。CrimeListActivity类会负责实现onCrimeUpdated(Crime)方法，从数据库获取并展示最新数据，重新加载CrimeListFragment的列表。

实现CrimeFragment的接口之前，先把CrimeListFragment.updateUI()方法修改为CrimeListActivity可以调用的公共方法，如代码清单17-10所示。

代码清单17-10　修改updateUI()方法的可见性（CrimeListFragment.java）

```
~~private~~ public void updateUI() {
    ...
}
```

然后，在CrimeFragment.java中，添加回调方法接口以及mCallbacks成员变量，并实现onAttach(...)方法和onDetach()方法，如代码清单17-11所示。

代码清单17-11　新增CrimeFragment回调接口（CrimeFragment.java）

```
private ImageButton mPhotoButton;
private ImageView mPhotoView;
private Callbacks mCallbacks;

/**
 * Required interface for hosting activities
 */
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}

public static CrimeFragment newInstance(UUID crimeId) {
    ...
}

@Override
public void onAttach(Context context) {
    super.onAttach(context);
    mCallbacks = (Callbacks) context;
}

@Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onPause() {
        ...
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mCallbacks = null;
    }
```

然后在CrimeListActivity类中实现CrimeFragment.Callbacks接口，以实现在onCrimeUpdated(Crime)方法中重新加载crime列表，如代码清单17-12所示。

代码清单17-12　刷新显示crime列表（CrimeListActivity.java）

```
public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks, CrimeFragment.Callbacks {
    ...
    public void onCrimeUpdated(Crime crime) {
        CrimeListFragment listFragment = (CrimeListFragment)
                getSupportFragmentManager()
                        .findFragmentById(R.id.fragment_container);
        listFragment.updateUI();
    }
}
```

托管CrimeFragment的所有activity都必须实现CrimeFragment.Callbacks接口。因而在CrimePagerActivity中提供一个空接口实现，如代码清单17-13所示。

代码清单17-13　空接口实现（CrimePagerActivity.java）

```
public class CrimePagerActivity extends AppCompatActivity
        implements CrimeFragment.Callbacks {
    ...
    @Override
    public void onCrimeUpdated(Crime crime) {

    }
}
```

CrimeFragment有两项重复性任务：在CrimeLab中保存mCrime，以及调用mCallbacks.onCrimeUpdated(Crime)。添加一个便利方法处理它们，如代码清单17-14所示。

代码清单17-14　新增updateCrime()方法（CrimeFragment.java）

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ...
}
```

```
    private void updateCrime() {
        CrimeLab.get(getActivity()).updateCrime(mCrime);
        mCallbacks.onCrimeUpdated(mCrime);
    }

    private void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
    }
```

在CrimeFragment.java中，如果Crime对象的标题或问题处理状态有变动，触发调用updateCrime()方法，如代码清单17-15所示。

代码清单17-15 调用onCrimeUpdated(Crime)方法（CrimeFragment.java）

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        mTitleField.addTextChangedListener(new TextWatcher() {
            ...
            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {
                mCrime.setTitle(s.toString());
                updateCrime();
            }
            ...
        });
        ...
        mSolvedCheckbox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                mCrime.setSolved(isChecked);
                updateCrime();
            }
        });
        ...
    }
```

在onActivityResult(...)方法中，Crime对象的记录日期、现场照片以及嫌疑人都有可能发生变化，因此，还需在该方法中调用updateCrime()方法。当前，crime现场照片以及嫌疑人并没有出现在列表项视图中，但并排的CrimeFragment视图应该显示了这些更新，如代码清单17-16所示。

代码清单17-16 再次调用updateCrime()方法（CrimeFragment.java）

```
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        ...
        if (requestCode == REQUEST_DATE) {
            Date date = (Date) data
                    .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
```

```
            updateCrime();
            updateDate();
        } else if (requestCode == REQUEST_CONTACT && data != null) {
            ...
            try {
                ...
                String suspect = c.getString(0);
                mCrime.setSuspect(suspect);
                updateCrime();
                mSuspectButton.setText(suspect);
            } finally {
                c.close();
            }
        } else if (requestCode == REQUEST_PHOTO) {
            ...
            getActivity().revokeUriPermission(uri,
                    intent.FlAG_GRANT_WRiTE_URi_PERMiSSiON);

            updateCrime();
            updatePhotoView();
        }
    }
```

在平板设备上运行CriminalIntent应用。确认当`CrimeFragment`视图中发生任何变化时，`RecyclerView`视图都能够刷新显示。然后，在手机上运行CriminalIntent应用，确认一切正常。

至此，CriminalIntent应用终于能同时支持手机和平板了。

17.3　深入学习：设备屏幕尺寸的确定

Android 3.2之前，屏幕大小修饰符是基于设备的屏幕大小来提供可选资源的。屏幕大小修饰符将不同的设备分成了四大类别：`small`、`normal`、`large`及`xlarge`。

表17-1展示了每个类别修饰符的最低屏幕大小。

表17-1　屏幕大小修饰符

名　　称	最低屏幕大小	名　　称	最低屏幕大小
`small`	320×426dp	`large`	480×640dp
`normal`	320×470dp	`xlarge`	720×960dp

顺应允许开发者测试设备尺寸的新修饰符的推出，屏幕大小修饰符已在Android 3.2中弃用。表17-2列出了新引入的修饰符。

表17-2　独立的屏幕尺寸修饰符

修饰符格式	描　　述
`wXXXdp`	有效宽度：宽度大于或等于XXX dp
`hXXXdp`	有效高度：高度大于或等于XXX dp
`swXXXdp`	最小宽度：宽度或高度（两者中最小的那个）大于或等于XXX dp

假设想指定某个布局仅适用于屏幕宽度至少为300dp的设备，可以使用宽度修饰符，并将布局文件放入res/layout-w300dp目录下（w代表屏幕宽度）。类似地，我们也可以使用“hXXXdp”修饰符（h代表屏幕高度）。

设备方向变换的话，设备的宽和高也会交换。为了确定某个具体的屏幕尺寸，我们可以使用`sw`（最小宽度）。`sw`指定了屏幕的最小规格尺寸。设备的方向会变，因此`sw`可以是最小宽度，也可以是最小高度。例如，如果屏幕尺寸为1024×800，那么`sw`值就是800；而如果屏幕尺寸为800×1024，那么`sw`值仍然是800。

17.4 挑战练习：添加滑动删除功能

为了改善用户体验，请为CriminalIntent应用的`RecyclerView`添加滑动删除功能。也就是说，用户向右滑动一下，就能删除一条crime记录。

为了实现这个功能，你需要使用`ItemTouchHelper`类（developer.android.com/reference/android/support/v7/widget/helper/ItemTouchHelper.html）。这个类提供了滑动删除实现，包含在`RecyclerView`支持库中。

第 18 章 应用本地化

CriminalIntent应用预计会很火，为了方便更多的用户使用，我们决定首先实施中文本地化。

本地化是一个基于设备语言设置，为应用提供合适资源的过程。本章为CriminalIntent应用提供中文版字符串资源。设备语言如果设置为中文，Android就会自动找到并使用相应的中文资源，如图18-1所示。



图18-1　中文版CriminalIntent应用

18.1　资源本地化

语言设置是设备配置的一部分（详见第3章的“设备配置与备选资源”小节）。和处理屏幕方向、屏幕尺寸以及其他配置因素改变一样，Android也提供了用于不同语言的配置修饰符。本地化处理因而变得简单：创建带目标语言配置修饰符的资源子目录，并放入备选资源。其余工作就交给Android资源系统自动处理。

在项目工具窗口中，右键点击res/values目录，选择New → Values resource file菜单项。文件名输入strings.xml，Source set选main，Directory name设置为values。然后，在Available qualifiers列表窗口，选中Locale，使用>>按钮把它移入Chosen qualifiers窗口，在Language列表窗口中选zh: Chinese，此时，右边的Specific Region Only窗口会自动选中Any Region，这就是我们想要的，所以无需更改。

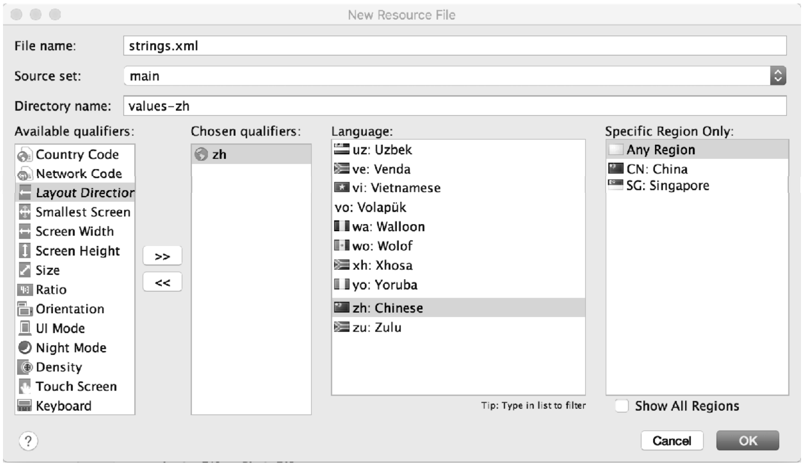


图18-2 新建资源文件

设置完成了。现在，新建资源文件窗口应该类似于图18-2。

注意，Android Studio会自动设置Directory name为values-zh。语言配置修饰符来自于ISO 639-1标准代码，每个修饰符都由两个字符组成。中文的修饰符为-zh。

点击OK按钮完成。带(zh)后缀的新strings.xml文件会在res/values下列出。现在观察一下，在项目工具窗口，strings资源文件都是按组归类的，如图18-3所示。

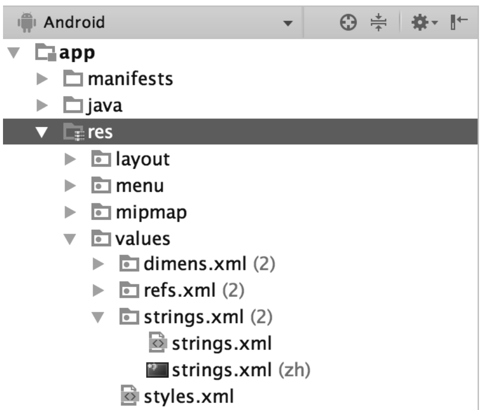


图18-3 新的strings.xml文件

然而，如果查看目录结构，你会看到项目现在有了另一个values目录：res/values-zh。新产生的strings.xml就放在这个目录里，如图18-4所示。



图18-4 在Project视图中查看新的strings.xml文件

现在，开始真正的中文定制。添加中文字符串资源给res/values-zh/strings.xml文件，如代码清单18-1所示。

代码清单18-1 添加中文备选字符串资源（res/values-zh/strings.xml）

```
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="crime_title_hint">crime简短描述</string>
    <string name="crime_title_label">标题</string>
    <string name="crime_details_label">明细</string>
    <string name="crime_solved_label">是否解决</string>
    <string name="date_picker_title">crime发生日期</string>
    <string name="new_crime">新增crime记录</string>
    <string name="show_subtitle">显示子标题</string>
    <string name="hide_subtitle">隐藏子标题</string>
    <string name="subtitle_format">%1$s crimes</string>
```

```
        <string name="crime_suspect_text">嫌疑人联系方式</string>
        <string name="crime_report_text">抗议或投诉</string>
        <string name="crime_report">%1$s!crime发生于 %2$s. %3$s, y %4$s</string>
        <string name="crime_report_solved">问题已解决</string>
        <string name="crime_report_unsolved">问题未解决</string>
        <string name="crime_report_no_suspect">没找到嫌疑人</string>
        <string name="crime_report_suspect">嫌疑人是 %s</string>
        <string name="crime_report_subject">crime处理情况报告</string>
        <string name="send_report">投诉方式</string>
    </resources>
```

这样便完成了为应用提供本地化资源的任务。要验证成果，请打开Settings，找到语言设置选项（Android版本繁多，但语言设置选项一般被标为Language and input、Language and Keyboard或其他类似名称），将设备语言改为简体中文。

运行CriminalIntent应用。真好，亲切又熟悉的中文界面出现了。

18.1.1　默认资源

英文语言的配置修饰符为-en。处理完中文本地化，你自然想到也把原来的values目录重命名为values-en。这可不是个好主意。现在假设你已经这么做了：应用现在有一个英文版values-en/strings.xml和一个中文版values-zh/strings.xml。

运行应用，一切都很正常，语言改为中文，也没问题。但是，如果有个用户把设备语言改为法语，会出现什么情况呢？问题来了，Android无法找到匹配当前语言设置的资源！后果很严重，有多严重？那要看应用的那些字符串资源是在哪里引用的。

如果不匹配的字符串资源用在XML布局文件中，应用会显示资源ID数值。若想亲眼见识一下，可在values/strings.xml文件中注释掉`crime_title_label`定义（如果想快速注释一行代码，可使用Command+/或Ctrl+/快捷键）。

代码清单18-2　注释掉`crime_title_label`定义（res/values/strings.xml）

```
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="crime_title_hint">Enter a title for the crime.</string>
    <!--<string name="crime_title_label">Title</string>-->
    ...
```

（fragment_crime.xml引用了`crime_title_label`）

```
<TextView
    style="?android:listSeparatorTextViewStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_title_label" />
```

在语言设置为英语的设备上运行应用。如图18-5所示，应用显示的不是标题文字，而是`crime_title_label`的资源ID。

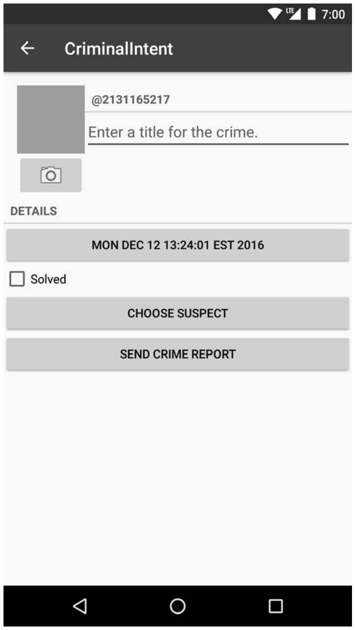


图18-5　布局找不到默认资源定义，转而使用资源ID

更糟的是，如果不匹配的字符串资源用在Java代码中，应用就会崩溃。在values/strings.xml文件中注释掉`crime_report_subject`定义，亲眼见证一下。

代码清单18-3　注释掉`crime_report_subject`定义（res/values/strings.xml）

```
<string name="crime_report_no_suspect">there is no suspect.</string>
<string name="crime_report_suspect">the suspect is %s.</string>
<!--<string name="crime_report_subject">CriminalIntent Crime Report</string>-->
```

(在CrimeFragment.java文件中，`crime_report_subject`是在crime report按钮监听器代码中引用的。)

```
mReportButton.setOnClicklistener(new View.OnClicklistener() {
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_SEND);
                i.setType("text/plain");
                i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
                i.putExtra(Intent.EXTRA_SUBJECT,
                        getString(R.string.crime_report_subject));
                i = Intent.createChooser(i, getString(R.string.send_report));
                startActivity(i);
            }
        });
```

运行应用，点SEND CRIME REPORT按钮。如图18-6所示，应用崩溃了。

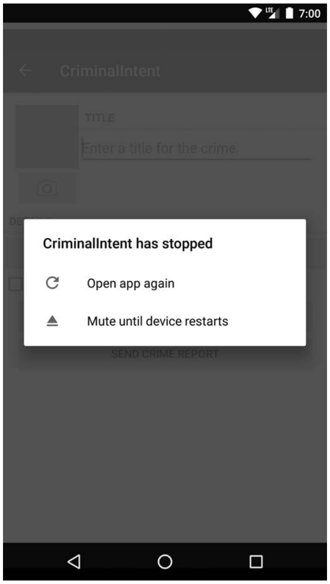


图18-6　Java代码找不到资源定义，应用崩溃

总结一下：提供默认资源非常重要。没有配置修饰符的资源就是Android的默认资源。如果无法找到匹配当前配置的资源，Android就会使用默认资源。默认资源至少能保证应用正常运行。

（暂时保留`crime_report_subject`和`crime_title_label`的定义注释，稍后会用到。）

例外的屏幕显示密度

Android默认资源使用规则并不适用于屏幕显示密度。项目的drawable目录通常按屏幕显示密度要求，带有-mdpi、-xxhdpi这样的修饰符。不过，Android决定使用哪一类drawable资源并不是简单地匹配设备的屏幕显示密度，也不是在没有匹配的资源时直接使用默认资源。

最终的选择取决于对屏幕尺寸和显示密度的综合考虑。Android甚至可能会选择低于或高于当前设备屏幕密度的drawable资源，然后通过缩放去适配设备。访问网页developer.android.com/guide/practices/screens_support.html，了解更多相关信息。无论如何，请记住一点：不要在res/drawable/目录下放置默认的drawable资源。

18.1.2　检查资源本地化完成情况

是否已为某种语言提供全部本地化资源？应用支持的语言越来越多，想快速确认也越来越难。Google早已想到这点，所以Android Studio提供了资源翻译编辑器这个工具。这个便利工具能集中化查看资源翻译完成情况。在项目工具窗口，右键点击某个语言版本的strings.xml，选择Open Translations Editor菜单项打开资源翻译编辑器。如图18-7所示，资源翻译编辑器随即显示了对应语言的全部资源定义。`crime_report_subject`和`crime_title_label`的定义已注释，所以它们被标红了。

18


Show only keys needing translations　　Order a translation...

Key	Default Value	Untra...	Chinese (zh)
app_name	CriminalIntent		CriminalIntent
crime_details_label	Details		明细
crime_report	%1$s![...]		%1$s![...]
crime_report_no_susp	there is no suspect.		没找到嫌疑人
crime_report_solved	The case is solved		问题已解决
crime_report_subject			
crime_report_suspect	the suspect is %s.		嫌疑人是 %s
crime_report_text	Send Crime Report		抗议或投诉
crime_report_unsolvec	The case is not solved		问题未解决
crime_solved_label	Solved		是否解决
crime_suspect_text	Choose Suspect		嫌疑人联系方式
crime_title_hint	Enter a title for the crime.		crime简短描述
crime_title_label			
date_picker_title	Date of crime:		crime发生日期
hide_subtitle	Hide Subtitle		隐藏子标题
new_crime	New Crime		新增crime记录
send_report	Send crime report via		发送方式
show_subtitle	Show Subtitle		显示子标题
subtitle_format	%1$d crimes		%1$d crimes

图18-7　检查应用本地化完成情况

（现在可以取消注释`crime_report_subject`和`crime_title_label`的定义了，别忘了用快捷键。）

18.1.3　区域修饰符

修饰资源目录也可以使用语言加区域修饰符，这样可以让资源使用更有针对性。例如，西班牙语可以使用-es-rES修饰符。其中，r代表区域，ES是西班牙语的ISO 3166-1-alpha-2标准码。配置修饰符对大小写不敏感。但最好遵守Android命名约定：语言代码小写，区域代码大写，但前面加个小写的r。

注意，语言区域修饰符，如-es-rES，看上去像两个不同的修饰符的合体，实际不是这样。这是因为，区域本身不能单独用作修饰符。

图18-8展示了Android不同系统版本的区域资源匹配策略。

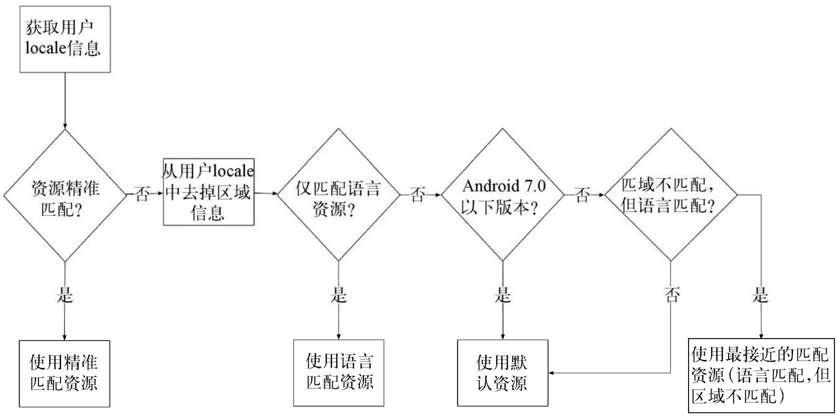


图18-8　区域资源匹配策略（Nougat及其之前的系统版本）

如果一个资源修饰符同时包含locale和区域，那么它有两次机会匹配用户的locale。首先，如

果语言和区域修饰同时匹配用户的locale，那这就是一次精准匹配。如果是非精准匹配，系统会去除区域修饰，然后仅以语言去做精准匹配。

在运行Nougat之前的系统版本的设备上，如果找不到匹配的资源，应用就会使用无任何修饰符的默认资源。Nougat系统已优化locale支持，支持更多locale以及支持同一设备选择多个locale。因此，为了让应用显示更准确的语言，系统使用了更智能化的资源匹配策略。如果找不到精准匹配，也找不到仅针对语言的匹配，系统就会去匹配有同样语言而区域不同的资源。

来看个例子。假设设备语言设置为西班牙语，区域设置为智利，如图18-9所示。应用准备了西班牙版本和墨西哥版本的资源（values-es-rES和values-es-rMX），以及默认的strings.xml资源。如果是Nougat之前的系统版本，应用会使用默认资源。如果是Nougat系统，应用会使用values-es-rMX/strings.xml资源。怎么样，智能多了吧！

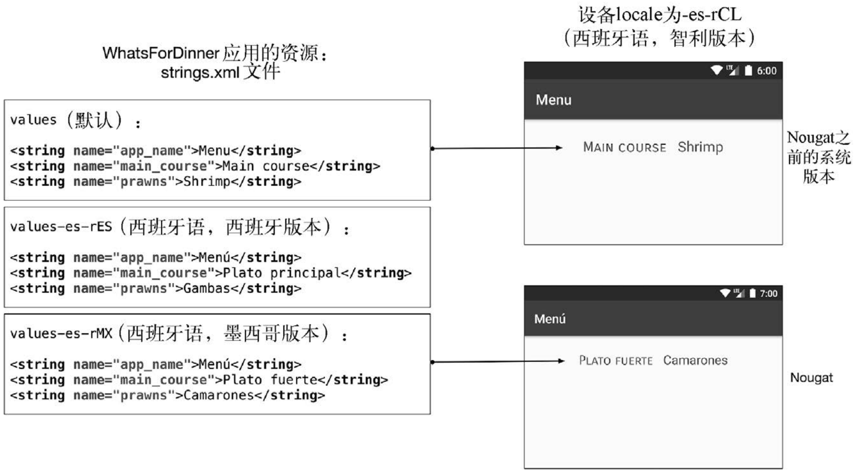


图18-9　区域匹配实例（Nougat及其之前的系统版本）

以上例子有策划之嫌。但无论如何，可从中得出一个重要结论：资源应尽可能通用，最好是使用仅限语言的修饰目录，尽量少用区域修饰。就上例来说，与其维护三类不同区域西班牙语的资源，不如只提供values-es版资源。这样，不仅方便开发维护，也方便适配不同版本的系统。

测试定制区域

不同设备不同版本的Android会支持不同的区域。有时，你也可能提供针对某个特定区域的资源，但测试时发现测试设备无此区域。碰到这事，不用担心，你可以使用模拟器上的定制区域工具。用这个工具，可以定制出系统镜像没有的区域。模拟器运行时，会模拟出一个包含语言和区域的运行时配置。这样，你就可以测试应用表现如何了。

模拟器已内置定制区域工具。运行模拟器，打开应用启动屏，点击定制区域应用图标启动它。启动之后，这个工具会列出系统现有的区域，并允许你添加新区域进行测试，如图18-10所示。

注意，如果定制了模拟器本身不支持的区域，系统界面仍然会使用默认语言。测试应用会使用所选定制区域匹配资源。

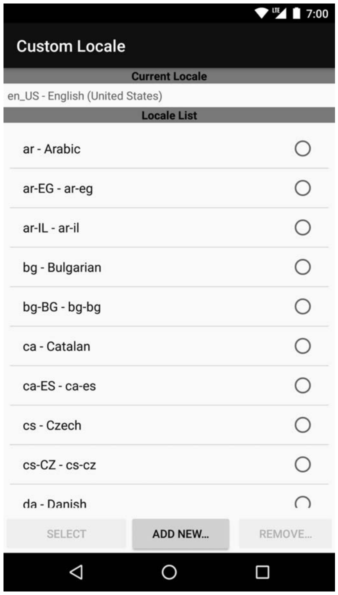


图18-10　定制区域工具

18.2　配置修饰符

目前为止，我们已见过好几个配置修饰符，它们都用于提供可选资源，如语言（values-zh/）、屏幕方位（layout-land/）、屏幕显示密度（drawable-mdpi/）以及屏幕尺寸（layout-sw600dp/）。

表18-1列出了一些设备配置特征。针对它们，Android提供配置修饰符以更好地匹配资源。

表18-1　可带配置修饰符的设备配置特征

1	移动国家码，通常附有移动网络码
2	语言代码，通常附有区域代码
3	布局方向
4	最小宽度
5	可用宽度
6	可用高度
7	屏幕尺寸
8	屏幕纵横比
9	圆形屏幕（API 23+）
10	屏幕方位
11	UI模式
12	夜间模式
13	屏幕显示密度

（续）

14	触摸屏类型
15	键盘可用性
16	首选输入法
17	导航键可用性
18	非文本导航方法
19	API级别

可访问网页developer.android.com/guide/topics/resources/providing-resources.html#Alternative-Resources，查看表中所有设备配置特征描述及其对应配置修饰符的使用例子。

旧系统版本Android并非支持所有配置修饰符。系统知道这一点，所以会给Android 1.0之后出现的修饰符加上平台版本修饰。例如，圆形屏幕修饰符自API 23级别引入，如果你使用它，系统会自动加上v23。因此，如果为新设备引入资源修饰符，根本不用担心在旧系统会遇到问题。

18.2.1　可用资源优先级排定

考虑到有那么多匹配资源的配置修饰符，有时，会出现设备配置与好几个可选资源都匹配的情况。遇到这种状况，Android会基于表18-1的顺序确定修饰符的使用优先级。

为了实际了解这种优先级排定，我们为CriminalIntent应用再添加一种可选资源：更详细的`crime_title_hint`字符串资源（针对屏幕宽度至少600dp的设备）。`crime_title_hint`资源显示在crime的可编辑标题框里（用户输入标题前）。应用运行在平板或者处于水平模式的设备上时（屏宽至少600dp），标题框才会显示更详细的内容，提示用户输入crime标题。

新建一个字符串资源文件，并将其放入values-w600dp目录。（-w600dp会匹配屏幕宽度大于600dp的任何设备。如果设备水平模式的屏幕符合条件也可以。）如何添加新资源文件可参考18.1节。设置完成后的界面大致如图18-11所示。

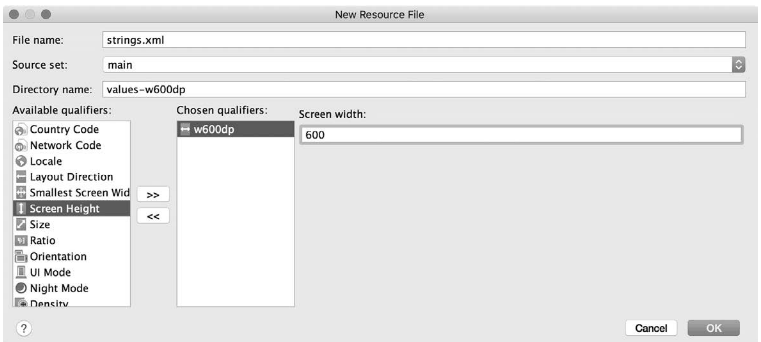


图18-11　为大屏添加字符串资源

打开values-w600dp/strings.xml文件，参照代码清单18-4，为`crime_title_hint`添加更详细的文字描述。

代码清单18-4　针对宽屏的字符串资源（values-w600dp/strings.xml）

```
<resources>
    <string name="crime_title_hint">
        Enter a meaningful, memorable title for the crime.
    </string>
</resources>
```

在宽屏上，我们只希望看到`crime_title_hint`字符串有不同的描述，所以添加这一个就可以了。字符串备选资源（也包括其他values资源）都是基于每一个字符串提供，因此，字符串资源相同时，无需再复制一份。重复的字符串资源只会导致未来的维护噩梦。

现在总共有三个版本的`crime_title_hint`资源：values/strings.xml文件中的默认版本、values-zh/strings.xml文件中的中文备选版本以及values-w600dp/strings.xml文件中的宽屏设备备选版本。

在设备语言设置为简体中文的前提下，运行CriminalIntent应用，然后旋转设备至水平模式。因为中文备选版本的资源优先级最高，所以我们看到的是来自于values-zh/strings.xml文件的字符串资源，如图18-12所示。



图18-12　Android排定语言优先级高于屏幕方位

也可以将设备的语言重新设置为英语，然后再次运行应用，确认宽屏模式的字符串资源能如期出现。

18.2.2　多重配置修饰符

可以在同一资源目录上使用多个配置修饰符。这需要各配置修饰符按照优先级别顺序排列。因此，values-zh-land是一个有效的资源目录名，而values-land-zh目录名则无效。（在新建资源文件对话框中，工具会自动配置正确的目录名。）

为CriminalIntent应用准备宽屏模式的中文字符串资源。创建的资源目录名应为values-zh-w600dp。参照代码清单18-5，打开values-zh-w600dp/strings.xml文件，为`crime_title_hint`添加中文字符串资源。

代码清单18-5　创建宽屏中文版字符串资源（values-zh-w600dp/strings.xml）

```
<resources>
    <string name="crime_title_hint">
        请输入简短、好记的crime描述
    </string>
</resources>
```

在设备语言已设置为简体中文的前提下，运行CriminalIntent应用，确认新的备选资源如期出现，如图18-13所示。



图18-13　宽屏模式下的中文字符串资源出现了

18.2.3　寻找最匹配的资源

我们来看看Android是如何确定`crime_title_hint`资源版本的。首先，当前设备有以下四个版本的字符串备选资源：

- ❑ values/strings.xml
- ❑ values-zh/strings.xml
- ❑ values-w600dp/strings.xml
- ❑ values-zh-w600dp/strings.xml

其次，在设备配置方面，有台Nexus 5X，语言设为简体中文，屏宽600dp以上（可用宽度731dp，可用高度411dp）。

1. 排除不兼容的目录

要找到最匹配的资源，Android首先排除不兼容当前设备配置的资源目录。

结合备选资源和设备配置来看，四个版本的备选资源均兼容设备的当前配置。（如果设备旋转至竖直模式，设备配置会改变。此时，values-w600dp/与values-zh-w600dp/资源目录不兼容当前配置，因此排除。）

2. 按优先级表排除不兼容的目录

筛掉不兼容的资源目录后，自优先级最高的MCC（移动国家码）开始，Android逐项查看并按优先级表继续筛查不兼容目录（表18-1）。如果有任何以MCC为修饰符的资源目录，那么所有

18

不带MCC修饰符的都会被排除。如果仍有多个目录匹配，Android就继续按次高优先级筛选，如此反复，直至找到唯一满足兼容性的目录。

本例中没有目录包含MCC修饰符，因此无法筛选掉任何目录。接着，Android查看到次高优先级的设备语言修饰符。values-zh/和values-zh-w600dp/目录包含语言修饰符。所以，不包含语言修饰符的values-w600dp/可排除。

由于仍有多个目录匹配，因此继续看优先级表，接下来是屏幕宽度。此时，Android会找到一个带屏宽修饰符的目录以及两个不带屏宽修饰符的目录，由此，values/和values-zh/目录也被排除。就这样，values-zh-w600dp/成了唯一满足兼容需求的目录。因而，Android最终确定使用values-zh-w600dp/目录下的资源。

18.3　测试备选资源

开发应用时，为了查看布局以及其他资源的使用效果，一定要针对不同设备配置做好测试。在虚拟设备或实体设备上测试都行，还可以使用图形布局工具测试。

图形布局工具有很多选项，用以预览布局在不同配置下的显示效果。这些选项有屏幕尺寸、设备类型、API级别以及设备语言等。

要查看这些选项，可在图形布局工具中打开fragment_crime.xml文件，试用如图18-14所示的工具栏上的一些选项设置。



图18-14　使用图形布局工具预览资源

如果想确认项目是否包括所有必需的默认资源，可设置设备使用未提供本地化资源的语言。运行应用，查看所有视图界面并旋转设备。如果应用崩溃，请查看LogCat中的“Resource not found...”错误信息，排查缺少哪些默认资源。也请关注是否存在非崩溃型问题，如前面提到的用户界面显示资源ID那样的问题。

恭喜！CriminalIntent应用支持中英文了。有了原生用户界面，应用用起来更高效。怎么样，让应用支持新语言很简单吧！也就是添加带修饰符的额外资源文件而已！

18.4　挑战练习：日期本地化

你可能已经注意到了，不管设备locale怎么调整，CriminalIntent应用的日期依然是美国格式。请按照设备locale设置，进一步本地化，让日期以中文年月日显示。这个练习应该难不倒你。

查阅开发者文档有关`DateFormat`类的用法和指导。`DateFormat`类有个日期格式化工具，支持按locale做日期格式化。使用该类内置的配置常量，还可以进一步定制日期显示。

第 19 章 Android辅助功能

本章，我们让CriminalIntent应用更易用。一个易用的应用适合所有人使用，即便是视力、行动、听力有障碍的人也能使用。这些障碍有永久性的，也有暂时性的或特定场景下的：刚经过眼科检查后，瞳孔放大，眼睛会看不清楚；做饭油乎乎的手难以触碰屏幕；音乐厅里的音乐声盖过了手机的一切声音，等等。总之，应用越易用，用户用起来就越开心。

开发适合所有人的易用应用非常困难，但不能因为困难就退缩。本章，利用学习开发或设计易用应用的突破口，我们先迈出一小步，让有视力障碍的用户也能方便地使用CriminalIntent应用。

本章不会修改用户界面。我们会使用一个叫TalkBack的辅助工具，让应用更加易用。

19.1 TalkBack

TalkBack是Google开发的Android屏幕阅读器。基于用户的操作，它能读出屏幕上的内容。

TalkBack实际是一个辅助服务，这个特别的部件能读取应用屏幕上的信息（无论哪种应用都可以）。只要不嫌麻烦，谁都可以设计开发这样的辅助服务。但TalkBack已经非常好用，其应用相当广泛。

要使用TalkBack，首先要有一台Android设备（虚拟设备不支持TalkBack）。确保手机没有静音。不过，建议先找副耳机戴着，因为一旦启用TalkBack，手机就喋喋不休地说个没完。

要启用TalkBack，请打开设置，点按辅助功能。在服务类别下，点按TalkBack打开它。然后，点按右上角开关启用TalkBack服务，如图19-1所示。

如图19-2所示，Android会弹出一个对话框，要求用户对诸如监测用户行为、修改某些设置这样的操作授权。点击OK按钮同意。

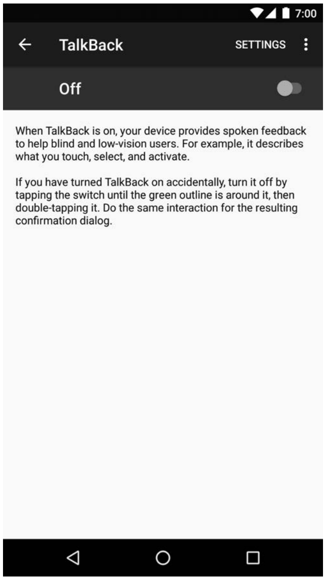


图19-1　TalkBack设置屏

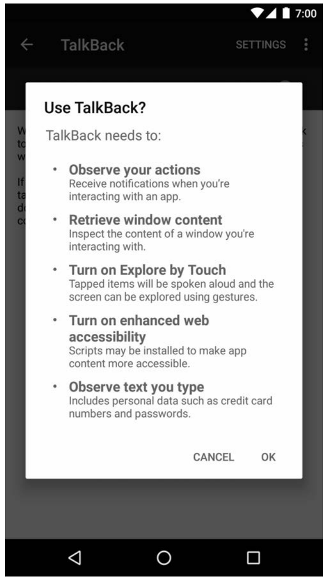


图19-2　给TalkBack授权

现在，TalkBack已处于启用状态（如果是首次使用，还会看到使用演示教程）。点按工具栏向上按钮退出。

注意，屏幕上是不是有了变化？一个绿框出现在向上按钮上，如图19-3所示。而且设备开始说话："向上导航按钮，连点两下可激活它。"



图19-3　TalkBack已启用

（注意，在移动设备屏幕上操作时，“点按”是常见的说法，但TalkBack却使用“点”以及“连点两下”。这并不常见。）

绿框表示当前UI元素获得了辅助焦点。一次只能有一个UI元素得到辅助焦点。UI元素得到辅助焦点后，TalkBack会提供该UI元素的信息。

设备启用TalkBack后，点操作会给予UI元素焦点，连点两下会激活。所以，当向上按钮获得焦点后，连点两下就回到上一屏了。如果是checkbox获得焦点，连点两下就是切换勾选状态。（同样，如果设备锁屏了，点锁屏按钮，然后连点两次屏幕任何地方就会解锁。）

19.1.1　点击浏览

只要启用了TalkBack，*点击浏览*（Explore by Touch）功能也会开启。这就意味着，点按某UI元素，设备就会读出相关信息。（当然，被点按的UI元素要有可读信息才行。）

让向上按钮仍处于聚焦状态，连点两次屏幕任何地方，TalkBack就会读出：“Accessibility。”

Android框架里的组件，如`Toolbar`、`RecyclerView`、`ListView`以及`Button`等，默认都支持TalkBack。想要用好TalkBack辅助功能，应尽可能多用框架内置组件。当然，也可以让定制组件支持TalkBack辅助功能，不过，这个话题比较复杂，已超出本书教学范畴。

需要两根手指按住屏幕上下滚动才能滚动列表。滚动时，设备会发出声响。这实际是对滚动的一种声音反馈。

19.1.2　线性浏览

想象一下，你第一次点击浏览一个应用会是什么情况？很可能你不知道点击哪个按钮才对。你明白，要想知道按了什么按钮，只能等TalkBack读出聚焦按钮说明才行。这会是一种什么体验？结果很可能是多次按同一个按钮，甚至是完全摸不着头脑。

好在TalkBack还有线性浏览功能。事实上，使用TalkBack最常见的方式就是使用线性浏览：右滑屏幕，辅助焦点移动到下一个UI元素；左滑，移动到上一个UI元素。这样，用户就可以线性浏览应用，再也不用碰运气了。

下面一起来体验一下。启动CriminalIntent应用，进入crime明细界面。点工具栏上的标题栏，让其聚焦。设备开始朗读“CriminalIntent”，如图19-4所示。

现在，向右滑屏。如图19-5所示，辅助焦点随即移动到添加新crime记录的按钮上。TalkBack读到：“添加新crime，连点两下激活，连点两下并按住是长按。”对于菜单项和按钮这样的框架组件，TalkBack会默认读出组件上显示的文字。添加新crime的按钮上没有文字，TalkBack会去找其他地方。在菜单项XML文件中，我们指定过标题信息（title），于是TalkBack就找到该信息读出。有时，TalkBack也能告诉用户某个组件接受什么操作，或这是什么组件。

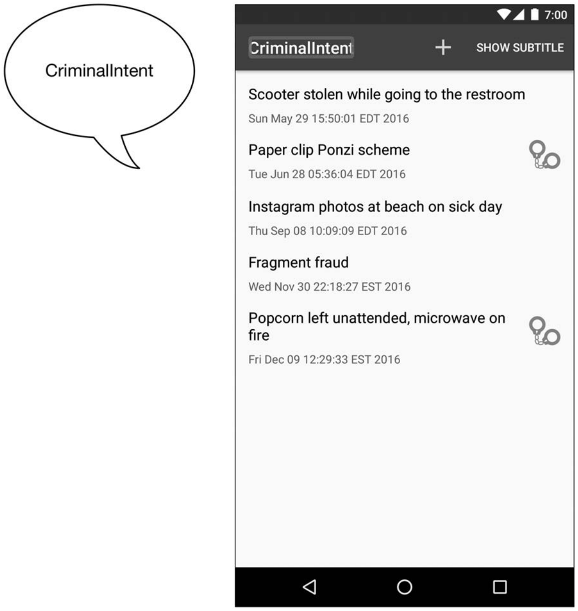


图19-4　标题已聚焦

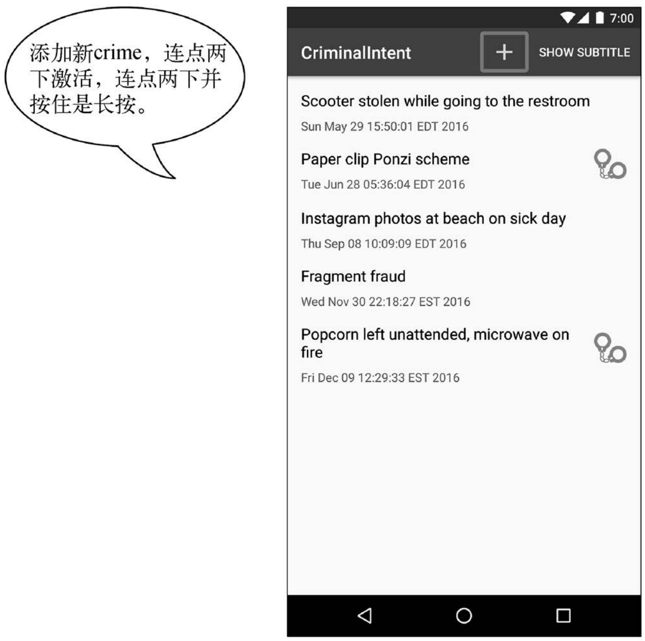


图19-5　添加新crime的按钮已聚焦

继续右滑，TalkBack给出SHOW SUBTITLE按钮的信息。再右滑，辅助焦点移动到第一条crime记录上。现在向左滑，辅助焦点又移回SHOW SUBTITLE按钮。总之，Android会智能有序地移动辅助焦点。这就是TalkBack的线性浏览。

19.2 实现非文字型元素可读

现在，点击工具栏上添加新crime的按钮，让其聚焦，等TalkBack读完按钮相关信息后，连点屏幕任意地方，进入crime明细页面。

19.2.1 添加内容描述

在crime明细页面，点击聚焦拍照按钮，如图19-6所示。TalkBack开腔了："按钮没文字，连点两下激活。"



图19-6 拍照按钮已聚焦

拍照按钮无文字描述，除了告诉用户连点两下激活，TalkBack没什么好说的。显然，这对于视力有障碍的人来说，并没有多大用处。

没关系，有办法解决。我们可以给`ImageButton`添加内容描述，这样TalkBack就有内容可读了。内容描述是一段针对组件的文字说明，供TalkBack朗读。（借处理拍照按钮的机会，一并给`ImageView`预览照片组件添加内容描述。）

要添加组件内容描述，可以在组件的布局XML文件里，添加`android:contentDescription`

属性。当然，也可以在布局实例化代码里，使用someView.setContentDescription(someString)方法。这两种方式，稍后都会尝试。

添加内容描述时，文字表述要简洁明了。TalkBack会朗读给用户听，虽然他们可以调快朗读速度，但简洁的文字能节约用户的时间。像组件是什么类型这种信息，TalkBack会自动提供，所以完全没必要写在内容描述里。

参照代码清单19-1，为ImageButton和ImageView添加内容描述。

代码清单19-1　添加内容描述字符串（res/values/strings.xml）

```
<resources>
    ...
    <string name="crime_details_label">Details</string>
    <string name="crime_solved_label">Solved</string>
    <string name="crime_photo_button_description">Take photo of crime scene</string>
    <string name="crime_photo_no_image_description">
        Crime scene photo (not set)
    </string>
    <string name="crime_photo_image_description">Crime scene photo (set)</string>
    ...
</resources>
```

然后，打开res/layout/fragment_crime.xml文件，参照代码清单19-2，给这两个组件添加android:contentDescription属性。

代码清单19-2　为ImageButton和ImageView添加内容描述（res/layout/fragment_crime.xml）

```
<ImageView
    android:id="@+id/crime_photo"
    android:layout_width="80dp"
    android:layout_height="80dp"
    android:background="@android:color/darker_gray"
    android:cropToPadding="true"
    android:scaleType="centerinside"
    android:contentDescription="@string/crime_photo_no_image_description" />

<ImageButton
    android:id="@+id/crime_camera"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:src="@android:drawable/ic_menu_camera"
    android:contentDescription="@string/crime_photo_button_description" />
```

运行CriminalIntent应用，聚焦拍照按钮，TalkBack开始朗读："陋习现场拍照按钮，连点两下激活。"这下，用户总算明白了。

接下来，点按照片预览处（当前显示的是灰色占位图）。你可能以为辅助聚焦框会上移，但绿色聚焦框包围了整个fragment视图区域，而并非移到ImageView组件上。TalkBack读的也是整个fragment视图的相关信息。问题在哪？

19.2.2 实现组件可聚焦

原来，ImageView组件没有做可聚焦登记。有些框架组件，如Button、CheckBox等，默认是可聚焦的；而像ImageView和TextView这样的框架组件需要手动登记。设置android:focusable属性值为true或使用监听器都可以让组件可聚焦。

如代码清单19-3所示，登记ImageView为可聚焦组件。

代码清单19-3 让ImageView可聚焦（res/layout/fragment_crime.xml）

```
<ImageView
  android:id="@+id/crime_photo"
  ...
  android:contentDescription="@string/crime_photo_no_image_description"
  android:focusable="true" />
```

运行CriminalIntent应用，点击crime缩略图组件，ImageView终于可聚焦了。如图19-7所示，TalkBack读道："crime陋习现场照片，当前未拍照。"



图19-7 可聚焦的ImageView

19.3 提升辅助体验

有些UI组件，如ImageView，虽然会给用户提供一些信息，但没有文字性内容。你也应该给这些组件添加内容描述。如果某个组件提供不了任何有意义的说明，应该把它的内容描述设置为null，让TalkBack直接忽略它。

你可能会认为，反正用户看不见，管它是不是图片，知道了又如何？这种想法不对。作为开发人员，理应让所有用户都能用到应用的全部功能，获得同样的信息。要有不同，那也是自身体验和使用方式上的差异。

好的辅助易用设计不是一字不漏地读屏幕。相反，应注重用户体验的一致性。重要的信息和上下文一定要全部传达。

现在，crime预览图就给了用户不好的体验。即使有照片，TalkBack也总说当前未拍照。现在我们一起感受一下。点按拍照按钮，然后连点屏幕两次激活，相机应用启动了，TalkBack说道："拍照。"点按并激活快门按钮拍摄一张照片。

确认所拍照片。（对于这个步骤，不同的相机应用可能有不同的操作。但不管怎样，都是先聚焦，再连点两下激活。）crime明细页面现在显示了刚拍的照片。聚焦ImageView视图，TalkBack读道："crime 陋习现场照片，当前未拍照。"

为了解决这个问题，让用户能听到正确信息，在updatePhotoView()方法里动态设置ImageView的内容描述，如代码清单19-4所示。

代码清单19-4　动态设置内容描述（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    ...
    private void updatePhotoView() {
        if (mPhotoFile == null || !mPhotoFile.exists()) {
            mPhotoView.setimageDrawable(null);
            mPhotoView.setContentDescription(
                    getString(R.string.crime_photo_no_image_description));
        } else {
            ...
            mPhotoView.setimageBitmap(bitmap);
            mPhotoView.setContentDescription(
                    getString(R.string.crime_photo_image_description));
        }
    }
}
```

现在，只要有照片更新，updatePhotoView()方法都会设置内容描述。如果mPhotoFile没值，内容描述会说明没拍照片，否则就明确说明已拍照。

运行CriminalIntent应用，查看刚拍过照的crime明细页面。如图19-8所示，点按图片聚焦，这次，TalkBack说道："crime 陋习现场照片，当前已拍照。"

使用 label 提供上下文

继续之前，给新crime加个标题。如图19-9所示，点按聚焦EditText框，TalkBack提示："编辑框，请给crime加标题。"



图19-8　带动态描述的可聚焦ImageView



图19-9　EditText提示

TalkBack默认读出`EditText`框里的内容。没输入标题之前，TalkBack会读出`android:hint`指定的内容。所以不需要也不应该给`EditText`设置内容描述。

然而，这实际是有问题的。眼见为实，在标题栏里输入文字：Sticker vandalism[1]。然后，点按聚焦`EditText`框，TalkBack提示：“编辑框，Sticker vandalism。”



图19-10　带crime标题的`EditText`

这个问题就是，如果输入了其他文字，TalkBack使用者就失去了上下文，不知道`EditText`框到底是做什么的。这对于视力好的人来说一目了然，因为上面有标题文字标签。如果就输入了简单标题，有视力障碍的用户就要费力猜测了。显然，使用体验就有了大差异。

可以很容易地标明`EditText`和`TextView`的关系，让TalkBack掌握同样的上下文关系。只要给`TextView`添加`android:labelFor`属性就可以了，如代码清单19-5所示。

代码清单19-5　给`EditText`打标签（res/layout/fragment_crime.xml）

```
<TextView
    style="?android:listSeparatorTextViewStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_title_label"
    android:labelFor="@+id/crime_title"/>
```

`android:labelFor`属性告诉TalkBack，`TextView`是以某个ID值指定的视图（`EditText`）的标签。`labelFor`定义在`View`类里，所以可以任意两个`View`互打标签。注意，这里必须使用`@+id`

① 意即“贴纸被故意破坏”。——编者注

这样的语法，因为我们正引用了一个当前还没定义的ID值。可以删除`EditText`定义中`android:id="@+id/crime_title"`这行代码里的+号，不过，留着也不会有问题。

运行应用。点按聚焦`EditText`框，TalkBack提示："编辑框，Sticker vandalism，用于crime标题。"

恭喜！现在，应用的易用性有了很大改善。很多开发人员总是找借口说，对辅助功能这块不熟悉，所以不愿为特殊人群提高应用的易用性。你看到了，让应用更好地支持TalkBack没那么难。而且，有了改善TalkBack的基础和经验，就更容易学会改善其他辅助功能，比如BrailleBack。

设计和实现具有辅助功能的应用容易让人望而却步。要知道，在这个领域，可是有好多专职工程师的。不过，与其害怕做不好而直接忽略，不如从基本做起：确保将屏幕上有意义的信息都传达给TalkBack用户；确保给予TalkBack用户充分的上下文信息。不要让他们浪费时间听废话。当然，最重要的是，倾听用户，虚心学习。

至此，CriminalIntent应用完成了。历经13章，我们创建了一个复杂的应用，它使用fragment，支持应用间通信，可以拍照，可以保存数据，甚至说中文。吃块蛋糕庆祝一下吧！

不过，吃完记得清理现场，好习惯要养成，人人都是监督者。

19.4　深入学习：使用辅助功能扫描器

本章，我们专注于让TalkBack用户更方便地使用应用。不过，这还没完，照顾视力障碍人群只是做了辅助工作的一小部分。

理论上，测试应用的辅助功能得靠真正每天在用辅助服务的用户。但即使现实不允许，也应竭尽所能。

为此，Google提供了一个辅助功能扫描器。它能评估应用在辅助功能方面做得如何并给出改进意见。现在拿CriminalIntent应用做个测试。

首先，访问play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor，按指导安装扫描器。

安装完成后，手机屏幕上会出现一个蓝色的打勾图标。好戏开始了，启动CriminalIntent应用，先不用管蓝色的打勾图标，直接进入应用的crime明细页面，如图19-11所示。

点按蓝色的打勾图标，辅助功能扫描器开始工作。分析时会看到进度条。一旦完成，会弹出一个窗口给出建议，如图19-12所示。



图19-11　启动CriminalIntent应用待分析

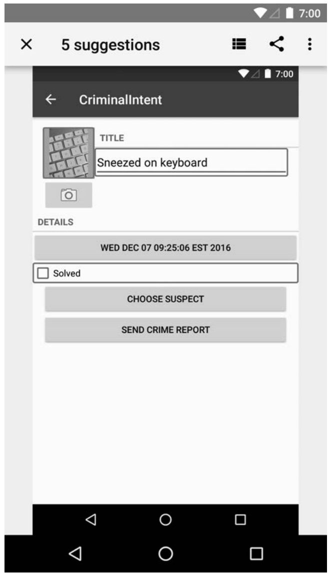


图19-12　辅助功能扫描器分析结果

可以看到ImageView、EditText和CheckBox都带框。这表明，扫描器认为这三个组件有潜在的辅助功能问题。点按EditText查看它的问题，如图19-13所示。

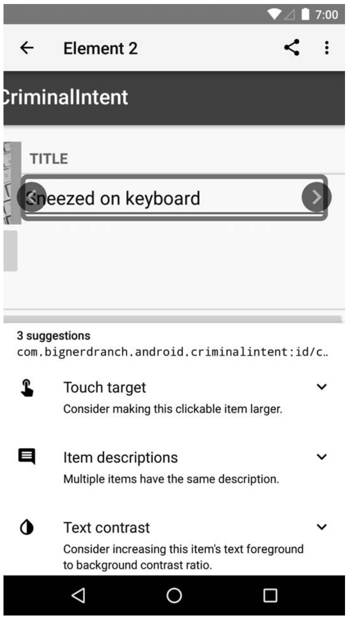


图19-13　EditText辅助功能改进意见

辅助功能扫描器有三个建议。第一个与EditText的尺寸有关。对所有触摸类组件，推荐的最小尺寸是48dp。EditText的高度不够，修改很容易，指定它的android:minHeight属性就可以了。

第二个和label有关。扫描器说，标题TextView和EditText提供的信息冗余，因为这两个组件和有关联。这里，扫描器说的冗余不是问题，忽略这个建议。

最后一个是关于文字颜色和其背景色的对比。有不少工具可以基于明亮度估测两种色彩的对比度。另外，也有一些网站，如randoma11y.com，按照易用对比度标准，列出很多颜色组合。这些标准是怎么订的呢？万维网联盟（一个为Web设计制定开放标准的国际社区）推荐，对于18 pt或更小的文字，至少要有4.5∶1的对比度。CriminalIntent应用的EditText的颜色对比度是3.52。要处理这个问题，调整其android:textColor属性和android:background属性就可以了。

想更深入了解扫描器的推荐，可点按右边的向下箭头查看细节，再点Learn More链接。

19.5 挑战练习：优化列表项

在crime列表页面，TalkBack会读出每条crime记录的标题和发生日期，但漏了crime是否已解决这一信息。给手铐图标添加内容描述，解决这个问题。

对于每条crime记录，TalkBack都要花点时间来读，这是因为日期格式那么长，而且是否已解决标志位于最右边。现在，再挑战一下自己，为屏幕上的每条记录都动态添加一个待读数据的汇总内容描述。

19.6 挑战练习：补全上下文信息

日期按钮和选择联系人按钮都有类似标题EditText的问题。无论是否使用TalkBack，用户都不是太明确按钮上是什么的日期。同样，选了联系人作为嫌疑人后，用户也可能不知道联系人按钮的作用是什么。用户也许能猜测出来，但为什么要让用户猜呢？

这就是设计的微妙之处。你或设计团队应该拿出最好的方案，平衡易用和简约的关系。

作为练习，请修改明细页面的设计，让用户充分把握数据和按钮间的上下文关系。和处理EditText标题一样，你可以为每个组件都添加label标签，或者完全重新设计明细页面。怎么做，自己选。总之，不要止步于眼前，要敢于对不好的说不，应用优化无止境。

19.7 挑战练习：事件主动通知

给ImageView添加动态内容描述后，crime缩略图组件的TalkBack体验获得极大改善。但是，TalkBack用户必须等点按并聚焦ImageView之后，才知道照片是否已拍或已更新。而视力正常的用户在从相机应用返回时就能看到照片更新情况。

你可以提供类似体验，让TalkBack用户在相机关闭时就能掌握照片更新情况。查阅文档研究一下View.announceForAccessibility(...)方法，看看怎么在CriminalIntent应用里使用。

或许你也考虑过在onActivityResult(...)方法里通知。若要这样做，会有与activity生命

周期相关的时间点掌控方面的问题要处理。不过，启动一个Runnable（详见第26章），做个延时处理可以绕开这个问题。以下给出参考代码：

```
mSomeView.postDelayed(new Runnable() {
    @Override
    public void run() {
        // Code for making announcement goes here
    }
}, SOME_DURATION_IN_MILLIS);
```

你也可以避开使用Runnable，想个办法确定发出通知的准确时间点。例如，可考虑在onResume()方法里通知。当然，前提是，你需要跟踪掌握用户是否刚从相机应用退出。

第 20 章 数据绑定与MVVM

开发GeoQuiz应用和CriminalIntent应用时，我们使用了Android标准工具：由Java对象组成的模型层、视图层级结构，以及控制器对象（activity和fragment）。对于这两个项目，MVC架构很适用。

本章，我们学习**数据绑定**（data binding）这个新工具。数据绑定只是一个工具，它不会告诉你怎么去用。不过，我们知道，所以，我们会教你如何在项目里使用它：实现Model-View-ViewModel（MVVM）新架构。此外，你还会学习使用**资源系统**（assets system）存储声音文件。

我们将在本章开始开发一个叫作BeatBox的新应用（如图20-1所示）。BeatBox不是传统意义上的音乐节拍盒，而是一个能帮你打败对手的神奇盒子。不过，它不会让你做出狂舞胳膊、致人受伤的危险事。实际上，它的作用出人意料：发出一种特殊的声响，直到对手求饶为止。



图20-1　本章要完成的BeatBox应用的效果

20.1　为何要用 MVVM 架构

目前为止，我们开发的应用都使用了简单版的MVC架构。而且，没出什么纰漏的话，应用都运行良好，让人挺有成就感。那么为何要改变？有什么问题吗？

本书实现的MVC架构比较适合小规模、简单型的应用。它方便开发人员理清项目结构，快速添加新功能，为开发打下坚实基础。应用因此得以快速完成并投入使用，在项目的早期阶段能保持稳定运行。

和所有项目一样，需求不断提出，应用一天比一天复杂。fragment和activity开始膨胀，逐渐变得难以理解和扩展。添加新功能或改bug需要耗费很长时间。这个时候，控制器层就需要做功能拆分了。

怎么拆？先搞清楚控制器类到底做了哪些工作，再把这些工作拆分到独立的小类里。让一个个拆开的小类协同工作。

那么，如何确定控制器类的不同使命呢？你的架构可以给你答案。人们给出高度概括的MVC（模型–视图–控制器）或者MVP（模型–视图–展示器）的说法，这些就是他们给这个问题的答案。无论如何，你的架构你自己最清楚，如何答题就看你了。

BeatBox应用使用MVVM架构。我们是MVVM架构的支持者。MVVM架构很好地把控制器里的臃肿代码抽到布局文件里，让开发人员很容易看出哪些是动态界面。同时，它也抽出部分动态控制器代码放入`ViewModel`类，这大大方便了开发测试和验证。

每个视图模型应控制成多大规模，这要具体情况具体分析。如果视图模型过大，你还可以继续拆分。总之，你的架构你把控。即使大家都用MVVM架构，业务不同，场景不一样，每个人的具体实现方法都有差异。

20.2　创建 BeatBox 应用

在Android Studio中，选择File → New → New Project...菜单项创建新项目。项目名称是BeatBox，公司域名是android.bignerdranch.com，最低SDK版本是API 19。新建一个名为`BeatBoxActivity`的空activity，其余默认项保持不变，完成项目创建。

BeatBox应用会用到`RecyclerView`。因此，打开项目设置添加com.android.support:recyclerview-v7依赖项。

在界面设计方面，应用主屏幕会显示一排排可以播放声音的按钮。因此，需要两个布局文件，分别用于网格和按钮。

首先创建`RecyclerView`布局文件。向导产生的res/layout/activity_beat_box.xml没用，因此直接改名为fragment_beat_box.xml。然后，参照代码清单20-1完成布局定义。

代码清单20-1　创建主布局文件（res/layout/fragment_beat_box.xml）

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
```

现在，在com.bignerdranch.android.beatbox包中，创建名为BeatBoxFragment的新Fragment，如代码清单20-2所示。

代码清单20-2 创建BeatBoxFragment（BeatBoxFragment.java）

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }
}
```

这个fragment实现先空着，稍后处理。

接下来，创建托管fragment的BeatBoxActivity。这里，BeatBoxActivity同样是直接继承CriminalIntent应用中用过的SingleFragmentActivity类。

首先，找到CriminalIntent项目的SingleFragmentActivity.java和activity_fragment.xml这两个文件，将其分别复制到com.bignerdranch.android.beatbox包和app/src/main/res/layout/目录。（从你自己的CriminalIntent应用文件夹或者随书文件中找到这些文件。）

然后，清空BeatBoxActivity类中的所有内容，改为继承SingleFragmentActivity类，并覆盖createFragment()方法，如代码清单20-3所示。

代码清单20-3 实现BeatBoxActivity（BeatBoxActivity.java）

```
public class BeatBoxActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return BeatBoxFragment.newInstance();
    }
}
```

简单的数据绑定

接下来的任务是实例化fragment_beat_box.xml并与RecyclerView关联起来。这类任务以前做过。这次，我们改用数据绑定来做，相比之前，会更方便快捷。

首先，在应用的build.gradle文件里启用数据绑定，如代码清单20-4所示。

代码清单20-4 启用数据绑定（app/build.gradle）

```
            versionCode 1
            versionName "1.0"
            testinstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
        }
        buildTypes {
            release {
                minifyEnabled false
                proguardFiles getDefaultProguardFile('proguard-android.txt'),
```

```
                'proguard-rules.pro'
        }
    }
    dataBinding {
        enabled = true
    }
}

dependencies {
```

这会打开IDE的整合功能，允许你使用数据绑定产生的类，并把它们整合到编译里去。

要在布局里使用数据绑定，首先要把一般布局改造为数据绑定布局。具体做法就是把整个布局定义放入<layout>标签，如代码清单20-5所示。

代码清单20-5　把一般布局改造为数据绑定布局（res/layout/fragment_beat_box.xml）

```
<layout
    xmlns:android="http://schemas.android.com/apk/res/android">
    <android.support.v7.widget.RecyclerView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</layout>
```

<layout>告诉数据绑定工具："这个布局由你来处理。"接到任务，数据绑定工具会帮你生成一个绑定类（binding class）。新产生的绑定类默认以布局文件命名。当然，不是snake_case这种格式，而是CamelCase格式（类名嘛）。

现在，fragment_beat_box.xml已经有了一个叫FragmentBeatBoxBinding的绑定类。这就是要用来做数据绑定的类：现在，实例化视图层级结构时，不再使用LayoutInflater，而是实例化FragmentBeatBoxBinding类。在一个叫作getRoot()的getter方法里，FragmentBeatBoxBinding引用着布局视图结构，而且也会引用那些在布局文件里以android:id标签引用的其他视图。

所以，FragmentBeatBoxBinding类有两个引用：getRoot()和recyclerView，前者指整个布局，后者指RecyclerView，如图20-2所示。

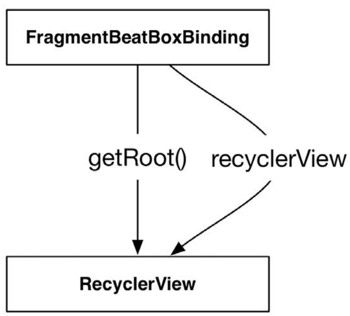


图20-2　FragmentBeatBoxBinding绑定类

你的布局只有一个视图，所以两个引用都指向了同一个视图：RecyclerView。

下面开始使用这个绑定类。在BeatBoxFragment里，覆盖onCreateView(...)方法，然后使用DataBindingUtil实例化FragmentBeatBoxBinding，如代码清单20-6所示。（你需要导入FragmentBeatBoxBinding类。如果Android Studio提示找不到，说明由于某种原因，FragmentBeatBoxBinding没有正常生成。请尝试重新编译项目，或重启Android Studio。）

代码清单20-6 实例化绑定类（BeatBoxFragment.java）

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newinstance() {
        return new BeatBoxFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        FragmentBeatBoxBinding binding = DataBindingUtil
                .inflate(inflater, R.layout.fragment_beat_box, container, false);

        return binding.getRoot();
    }
}
```

实例化绑定类后，就可以获取并配置RecyclerView了。

代码清单20-7 配置RecyclerView（BeatBoxFragment.java）

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newinstance() {
        return new BeatBoxFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        FragmentBeatBoxBinding binding = DataBindingUtil
                .inflate(inflater, R.layout.fragment_beat_box, container, false);

        binding.recyclerView.setLayoutManager(new GridLayoutManager(getActivity(),
                                                                     3));

        return binding.getRoot();
    }
}
```

数据绑定完成了，这就是我们所说的简单的数据绑定：不用findViewById(...)方法，转而使用数据绑定获取视图。稍后，我们还会学习数据绑定的高级用法。

接下来，创建按钮布局文件res/layout/list_item_sound.xml。这个布局也使用数据绑定，所以同样添加<layout>标签，如代码清单20-8所示。

代码清单20-8　创建声音布局文件（res/layout/list_item_sound.xml）

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <Button
        android:layout_width="match_parent"
        android:layout_height="120dp"
        tools:text="Sound name"/>
</layout>
```

接下来，创建一个使用list_item_sound.xml布局的SoundHolder，如代码清单20-9所示。

代码清单20-9　创建SoundHolder（BeatBoxFragment.java）

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newinstance() {
        return new BeatBoxFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...
    }

    private class SoundHolder extends RecyclerView.ViewHolder {
        private ListItemSoundBinding mBinding;

        private SoundHolder(ListItemSoundBinding binding) {
            super(binding.getRoot());
            mBinding = binding;
        }
    }
}
```

SoundHolder要使用刚才数据绑定工具已产生的绑定类：ListItemSoundBinding。

接着，创建一个关联SoundHolder的Adapter，如代码清单20-10所示。（不要输入任何方法，把光标移到RecyclerView.Adapter上，按Option+Return或Alt+Enter组合键，Android Studio会自动完成大部分代码。）

代码清单20-10　创建SoundAdapter（BeatBoxFragment.java）

```
public class BeatBoxFragment extends Fragment {
    ...
    private class SoundHolder extends RecyclerView.ViewHolder {
        ...
    }

    private class SoundAdapter extends RecyclerView.Adapter<SoundHolder> {
        @Override
        public SoundHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            LayoutInflater inflater = LayoutInflater.from(getActivity());
            ListItemSoundBinding binding = DataBindingUtil
```

```
                .inflate(inflater, R.layout.list_item_sound, parent, false);
            return new SoundHolder(binding);
        }

        @Override
        public void onBindViewHolder(SoundHolder holder, int position) {

        }

        @Override
        public int getItemCount() {
            return 0;
        }
    }
}
```

现在，在onCreateView(...)方法中使用SoundAdapter，如代码清单20-11所示。

代码清单20-11　使用SoundAdapter（BeatBoxFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    FragmentBeatBoxBinding binding = DataBindingUtil
            .inflate(inflater, R.layout.fragment_beat_box, container, false);

    binding.recyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));
    binding.recyclerView.setAdapter(new SoundAdapter());

    return binding.getRoot();
}
```

20.3　导入 assets

现在，需要把声音文件添加到项目里，以便应用调用。不过，这里不打算用资源系统，我们改用assets打包声音文件。可以把assets想象为经过精简的资源：它们也像资源那样打入APK包，但不需要配置系统工具管理。

使用assets有两面性：一方面，无需配置管理，可以随意命名assets，并按自己的文件结构组织它们；另一方面，没有配置管理，无法自动响应屏幕显示密度、语言这样的设备配置变更，自然也就无法在布局或其他资源里自动使用它们了。

所以，总体上讲，资源系统是更好的选择。然而，如果只想在代码中直接调用文件，那么assets就有优势了。大多数游戏就是使用assets加载大量图片和声音资源，BeatBox也这样。

现在我们来导入assets。首先创建assets目录。右键单击app模块，选择New → Folder → Assets Folder菜单项，弹出如图20-3所示的画面。不勾选Change Folder Location选项，保持Target Source Set的main选项不变，单击Finish按钮完成。

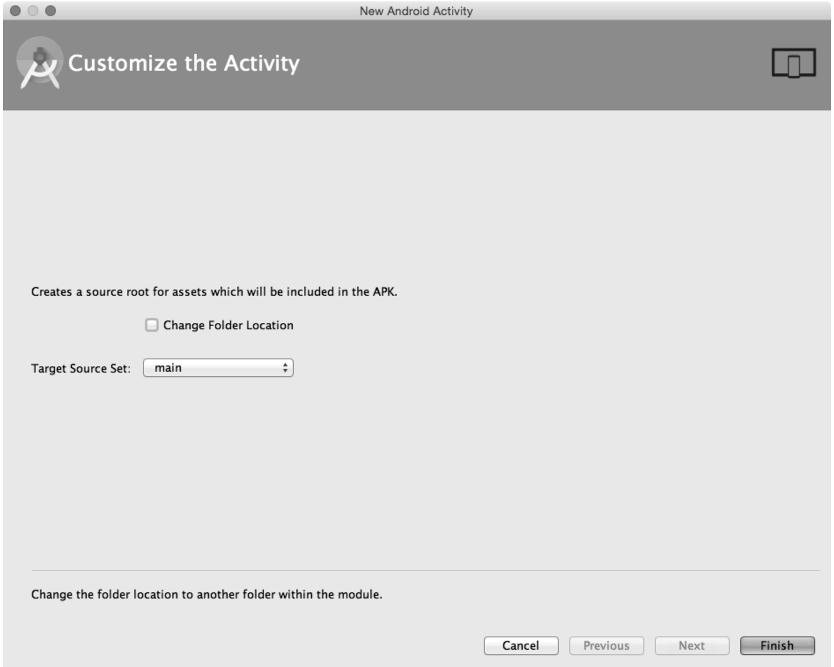


图20-3　创建assets目录

接着，右键单击assets目录，选择New → Directory菜单项，为声音资源创建sample_sounds子目录，如图20-4所示。

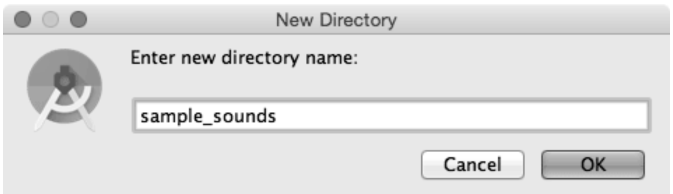


图20-4　创建sample_sounds子目录

assets目录中的所有文件都会随应用打包。为了方便组织文件，我们创建了sample_sounds子目录。与资源不同，子目录不是必需的，这里是为了组织声音文件。

声音文件哪里找呢？在www.freesound.org网站。plagasul是这个网站的用户，基于创作共用许可，他发布了一套声音文件（www.freesound.org/people/plagasul/packs/3/）。我们已重新打包提供。请使用以下地址获取：https://www.bignerdranch.com/solutions/sample_sounds.zip。

下载并解压缩文件至assets/sample_sounds目录，如图20-5所示。

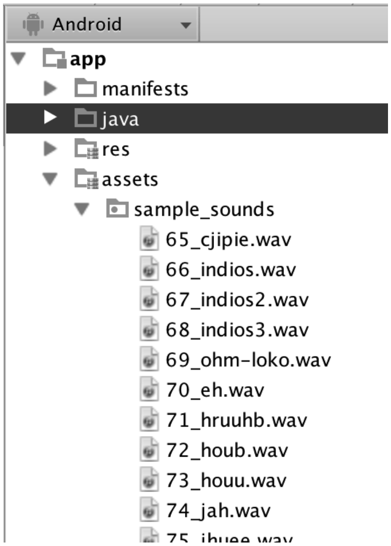


图20-5　已导入的assets

（确保这里只有.wav文件，而非尚未解压的.zip文件。）

重新编译应用，确保一切正常。接下来编写代码，列出所有这些资源并展示给用户。

20.4　处理 assets

assets导入后，还要能在应用中进行定位、管理记录以及播放。这需要新建一个名为BeatBox的资源管理类。如代码清单20-12所示，在com.bignerdranch.android.beatbox包中创建这个类，并添加两个常量：一个用于日志记录，另一个用于存储声音资源文件目录名。

代码清单20-12　创建BeatBox类（BeatBox.java）

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";
}
```

使用AssetManager类访问assets。可以从Context中获取它。既然BeatBox需要，不妨添加一个带Context参数的构造函数获取并留存它，如代码清单20-13所示。

代码清单20-13　获取AssetManager备用（BeatBox.java）

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";

    private AssetManager mAssets;
```

```
    public BeatBox(Context context) {
        mAssets = context.getAssets();
    }
}
```

通常，在访问assets时，可以不用关心究竟使用哪个Context对象。这是因为，在实践中的任何场景下，所有Context中的AssetManager都管理着同一套assets资源。

要取得assets中的资源清单，可以使用list(String)方法。如代码清单20-14所示，实现loadSounds()方法，调用它给出声音文件清单。

代码清单20-14　查看assets资源（BeatBox.java）

```
public BeatBox(Context context) {
    mAssets = context.getAssets();
    loadSounds();
}

private void loadSounds() {
    String[] soundNames;
    try {
        soundNames = mAssets.list(SOUNDS_FOLDER);
        Log.i(TAG, "Found " + soundNames.length + " sounds");
    } catch (IOException ioe) {
        Log.e(TAG, "Could not list assets", ioe);
        return;
    }
}
```

AssetManager.list(String)方法能列出指定目录下的所有文件名。因此，只要传入声音资源所在的目录，就能看到其中的所有.wav文件。

为了验证代码逻辑，在BeatBoxFragment中创建BeatBox实例，如代码清单20-15所示。

代码清单20-15　创建BeatBox实例（BeatBoxFragment.java）

```
public class BeatBoxFragment extends Fragment {

    private BeatBox mBeatBox;

    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mBeatBox = new BeatBox(getActivity());
    }
    ...
}
```

运行BeatBox应用。查看日志输出，看看列出了多少个声音文件。随书文件中提供了22个.wav文件，你应该看到如下结果。

```
…1823-1823/com.bignerdranch.android.beatbox I/BeatBox: Found 22 sounds
```


20.5　使用 assets

获取到资源文件名之后，要显示给用户看，最终还需要播放这些声音文件。所以，需要创建一个对象，让它管理资源文件名、用户应该看到的文件名以及其他一些相关信息。

创建一个这样的Sound管理类，如代码清单20-16所示。（别忘了，Android Studio可自动生成getter方法。）

代码清单20-16　创建Sound对象（Sound.java）

```
public class Sound {
    private String mAssetPath;
    private String mName;

    public Sound(String assetPath) {
        mAssetPath = assetPath;
        String[] components = assetPath.split("/");
        String filename = components[components.length - 1];
        mName = filename.replace(".wav", "");
    }

    public String getAssetPath() {
        return mAssetPath;
    }

    public String getName() {
        return mName;
    }
}
```

为了有效显示声音文件名，在构造方法中对其做一下处理。首先使用String.split(String)方法分离出文件名，再使用String.replace(String, String)方法删除.wav后缀。

接下来，在BeatBox.loadSounds()方法中创建一个Sound列表，如代码清单20-17所示。

代码清单20-17　创建Sound列表（BeatBox.java）

```
public class BeatBox {
    ...
    private AssetManager mAssets;
    private List<Sound> mSounds = new ArrayList<>();

    public BeatBox(Context context) {
        ...
    }

    private void loadSounds() {
```

```
        String[] soundNames;
        try {
            ...
        } catch (IOException ioe) {
            ...
        }

        for (String filename : soundNames) {
            String assetPath = SOUNDS_FOLDER + "/" + filename;
            Sound sound = new Sound(assetPath);
            mSounds.add(sound);
        }
    }

    public List<Sound> getSounds() {
        return mSounds;
    }
}
```

再让SoundAdapter与Sound列表关联起来，如代码清单20-18所示。

代码清单20-18　绑定Sound列表（BeatBoxFragment.java）

```
private class SoundAdapter extends RecyclerView.Adapter<SoundHolder> {
    private List<Sound> mSounds;

    public SoundAdapter(List<Sound> sounds) {
        mSounds = sounds;
    }
    ...
    @Override
    public void onBindViewHolder(SoundHolder soundHolder, int position) {
    }

    @Override
    public int getItemCount() {
        return 0;
        return mSounds.size();
    }
}
```

最后，在onCreateView(...)方法中传入BeatBox声音资源，如代码清单20-19所示。

代码清单20-19　传入声音资源（BeatBoxFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    FragmentBeatBoxBinding binding = DataBindingUtil
            .inflate(inflater, R.layout.fragment_beat_box, container, false);

    binding.recyclerView.setlayoutManager(new GridlayoutManager(getActivity(), 3));
    binding.recyclerView.setAdapter(new SoundAdapter());
    binding.recyclerView.setAdapter(new SoundAdapter(mBeatBox.getSounds()));
```

```
    return binding.getRoot();
}
```

现在，运行BeatBox应用，可以看到满是按钮的网格出现了，如图20-6所示。

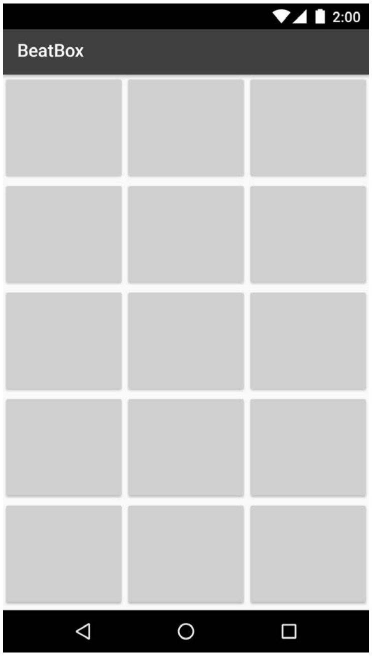


图20-6　空荡荡的按钮

要显示按钮文字，还需要使用新的数据绑定小工具。

20.6　绑定数据

使用数据绑定，我们还可以在布局文件中声明数据对象：

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="crime"
            type="com.bignerdranch.android.criminalintent.Crime"/>
    </data>
    ...
</layout>
```

然后，使用绑定操作符@{}就可以在布局文件中直接使用这些数据对象的值：

```
<CheckBox
    android:id="@+id/list_item_crime_solved_check_box"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:checked="@{crime.isSolved()}"
    android:padding="4dp"/>
```

在对象关系图中，可以这样表示（图20-7）：

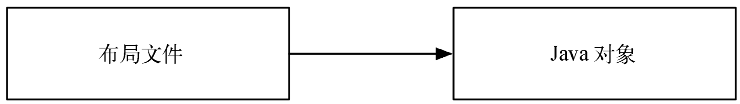


图20-7　绑定关系

我们的目标是在按钮上显示声音文件名。使用数据绑定，最直接的方式就是绑定list_item_sound.xml布局文件中的Sound对象，如图20-8所示。

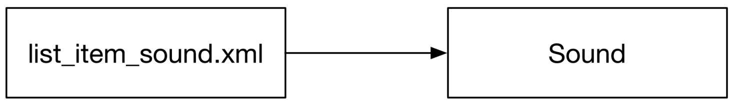


图20-8　直接绑定

然而，这似乎有架构问题。首先从MVC视角看看问题在哪，如图20-9所示。

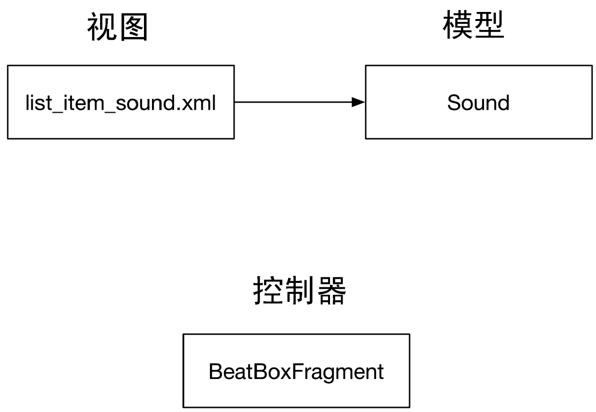


图20-9　断裂的MVC

不管是哪种架构，有一个指导原则都一样：责任单一性原则。也就是说，每个类应该只负责一件事情。按此原则，MVC是这样落实的：模型表明应用是如何工作的；控制器决定如何显示应用；视图显示你想看到的结果。

使用如图20-8所示的数据绑定，就破坏了责任划分。这是因为Sound模型对象不可避免地需要关心显示问题。代码也就此开始混乱了。模型层代码和控制器层代码里都是Sound.java。

为了避免Sound这样破坏单一性原则的情况，我们引入一种叫作视图模型的新对象（配合数据绑定使用）。视图模型负责如何显示视图，如图20-10所示。

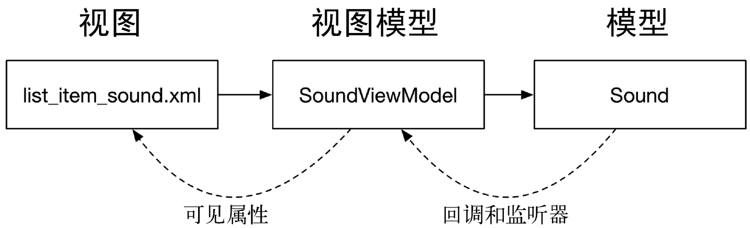


图20-10 MVVM

这种架构称为MVVM。从前控制器对象格式化视图数据的工作就转给了视图模型对象。现在，使用数据绑定，组件关联数据就能直接在布局文件里处理了。控制器对象（activity或fragment）开始负责初始化布局绑定类和视图模型对象，同时也是它们之间的联系纽带。

20.6.1 创建视图模型

首先来创建视图模型类。创建一个名为SoundViewModel的新类，然后添加两个属性：一个Sound对象，一个播放声音文件的BeatBox对象，如代码清单20-20所示。

代码清单20-20 创建SoundViewModel类（SoundViewModel.java）

```
public class SoundViewModel {
    private Sound mSound;
    private BeatBox mBeatBox;

    public SoundViewModel(BeatBox beatBox) {
        mBeatBox = beatBox;
    }

    public Sound getSound() {
        return mSound;
    }

    public void setSound(Sound sound) {
        mSound = sound;
    }
}
```

新添加的属性是adapter要用到的接口。对于布局，还需要一个额外的方法获取按钮要用的文件名。如代码清单20-21所示，加上这个方法。

代码清单20-21 添加绑定方法（SoundViewModel.java）

```
public class SoundViewModel {
    private Sound mSound;
    private BeatBox mBeatBox;
```

```
    public SoundViewModel(BeatBox beatBox) {
        mBeatBox = beatBox;
    }

    public String getTitle() {
        return mSound.getName();
    }

    public Sound getSound() {
        return mSound;
```

20.6.2　绑定至视图模型

现在，把视图模型整合到布局文件里。第一步是在布局文件里声明属性，如代码清单20-22所示。

代码清单20-22　声明视图模型属性（list_item_sound.xml）

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="viewModel"
            type="com.bignerdranch.android.beatbox.SoundViewModel"/>
    </data>
    <Button
```

这在绑定类上定义了一个叫viewModel的属性，同时还包括getter方法和setter方法。在绑定类里，可以用绑定表达式使用viewModel。

代码清单20-23　绑定按钮文件名（list_item_sound.xml）

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="viewModel"
            type="com.bignerdranch.android.beatbox.SoundViewModel"/>
    </data>
    <Button
        android:layout_width="match_parent"
        android:layout_height="120dp"
        android:text="@{viewModel.title}"
        tools:text="Sound name"/>
</layout>
```

在绑定表达式里，可以写一些简单的Java表达式，如链式方法调用、数学计算等。另外，也可以吃几颗“语法糖”。例如，上述viewModel.title实际就是viewModel.getTitle()的简写形式。数据绑定知道怎么帮你翻译。

最后一步就是关联使用视图模型。创建一个SoundViewModel，把它添加给绑定类，然后在SoundHolder里添加一个绑定方法，如代码清单20-24所示。

代码清单20-24 关联使用视图模型（BeatBoxFragment.java）

```
private class SoundHolder extends RecyclerView.ViewHolder {
    private ListItemSoundBinding mBinding;

    private SoundHolder(ListItemSoundBinding binding) {
        super(binding.getRoot());
        mBinding = binding;
        mBinding.setViewModel(new SoundViewModel(mBeatBox));
    }

    public void bind(Sound sound) {
        mBinding.getViewModel().setSound(sound);
        mBinding.executePendingBindings();
    }
}
```

在SoundHolder构造方法里，我们创建并添加了一个视图模型。然后，在绑定方法里，更新视图模型要用到的数据。

一般不需要调用executePendingBindings()方法。然而在这里，我们正在RecyclerView里更新绑定数据。考虑到RecyclerView刷新视图极快，我们迫使布局立即刷新。这样，RecyclerView的表现就更为流畅。

最后，实现onBindViewHolder(...)方法以使用视图模型，如代码清单20-25所示。

代码清单20-25 调用bind(Sound)方法（BeatBoxFragment.java）

```
        return new SoundHolder(binding);
    }

    @Override
    public void onBindViewHolder(SoundHolder holder, int position) {
        Sound sound = mSounds.get(position);
        holder.bind(sound);
    }

    @Override
    public int getItemCount() {
        return mSounds.size();
    }
```

运行应用，如图20-11所示，按钮终于显示文件名了。



图20-11　按钮现在充实了

20.6.3　绑定数据观察

一切看上去很美，不过这只是表面。如图20-12所示，如果向下滚动按钮网格，问题就出现了。



图20-12　问题暴露了

看到最下面一个按钮上的“67_INDIOS2”了吗？上面也有一个。上下反复滚动几次，还能看到其他重复的文件名。看样子是随机重复。

在SoundHolder.bind(Sound)方法里，我们更新了SoundViewModel的Sound，但布局不知道。而且，从图20-10可知，视图模型并不会给布局反馈信息，所以出现了上面的问题。

现在任务明确了，我们需要让它们沟通起来。这需要视图模型实现数据绑定的Observable接口。这个接口可以让绑定类在视图模型上设置监听器。这样，只要视图模型有变化，绑定类立即会接到回调。

实现这个接口理论上可行，但工作量太大。有没有其他好办法呢？答案是肯定的。现在就一起来看个聪明的做法（使用数据绑定的BaseObservable类）。

使用BaseObservable类需要三个步骤：

(1) 在视图模型里继承BaseObservable类；

(2) 使用@Bindable注解视图模型里可绑定的属性；

(3) 每次可绑定的属性值改变时，就调用notifyChange()方法或notifyPropertyChanged(int)方法。

在SoundViewModel里，让它继承BaseObservable类，注解可绑定的属性并调用notifyChange()方法，如代码清单20-26所示。

代码清单20-26　几行代码就搞定（SoundViewModel.java）

```
public class SoundViewModel extends BaseObservable {
    private Sound mSound;
    private BeatBox mBeatBox;

    public SoundViewModel(BeatBox beatBox) {
        mBeatBox = beatBox;
    }

    @Bindable
    public String getTitle() {
        return mSound.getName();
    }

    public Sound getSound() {
        return mSound;
    }

    public void setSound(Sound sound) {
        mSound = sound;
        notifyChange();
    }
}
```

这里，调用notifyChange()方法，就是通知绑定类，视图模型对象上所有可绑定属性都已更新。据此，绑定类会再次运行绑定表达式更新视图数据。所以，setSound(Sound)方法一被调用，ListItemSoundBinding就立即知道，并调用list_item_sound.xml布局里指定的Button.setText(String)方法。

上面，我们提到过另一个方法：notifyPropertyChanged(int)。这个方法和notifyChange()方法做同样的事，但覆盖面不一样。调用notifyChange()方法，相当于是说："所有的可绑定属

性都变了，请全部更新。”调用`notifyPropertyChanged(int)`方法，相当于是说：“只有`getTitle()`方法的值有变化。”



图20-13　这下完美了

再次运行应用。这次，无论怎么滚上滚下，一切都没问题了。

20.7　访问 assets

本章的工作全部完成了。下一章还会继续完善BeatBox应用，播放assets声音资源。

继续学习之前，一起花点时间深入探讨一下assets的工作原理。

`Sound`对象定义了assets文件路径。尝试使用`File`对象打开资源文件是行不通的。正确的方式是使用`AssetManager`：

```
String assetPath = sound.getAssetPath();
InputStream soundData = mAssets.open(assetPath);
```

这样才能得到标准的`InputStream`数据流。随后，和Java中的其他`InputStream`一样，该怎么用就怎么用。

不过，有些API可能还需要`FileDescriptor`。（下一章的`SoundPool`类就会用到。）这也好办，改用`AssetManager.openFd(String)`方法就行了。

```
String assetPath = sound.getAssetPath();
// AssetFileDescriptors are different from FileDescriptors,
AssetFileDescriptor assetFd = mAssets.openFd(assetPath);
// but you get can a regular FileDescriptor easily if you need to.
FileDescriptor fd = assetFd.getFileDescriptor();
```

20.8 深入学习：数据绑定再探

数据绑定可学的还有很多，本书无法全部覆盖。不过，如果你有兴趣，不妨再多了解一下。

20.8.1 lambda 表达式

在布局文件里，还可以使用lambda表达式写点短回调。以下是一些简化版的Java lambda表达式：

```
<Button
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:text="@{viewModel.title}"
    android:onClick="@{(view) -> viewModel.onButtonClick()}"
    tools:text="Sound name"/>
```

和Java 8 lambda表达式差不多，上述表达式会转成目标接口实现（这里是`View.OnClickListener`）。和Java 8 lambda表达式不同的是，这些表达式的语法有些特殊：参数必须在括号里，最右边一定要有一个表达式。

另外，还有一点和Java 8 lambda表达式不同：如果用不到，lambda参数可以不写。所以，下面这个写法也可以。

```
android:onClick="@{() -> viewModel.onButtonClick()}"
```

20.8.2 更多语法糖

数据绑定还有一些方便的语法可用。最方便的一个是使用单引号代替双引号：

```
android:text="@{`File name: ` + viewModel.title}"
```

这里，`` `File name: ` ``和`"File name: "`是一样的。

绑定表达式也有一个遇`null`值就合并的操作符：

```
android:text="@{`File name: ` + viewModel.title ?? `No file`}"
```

如果`title`是`null`，`??`操作符就返回`"No file"`值。

此外，数据绑定还有`null`自动处理机制。在上面的代码中，即使`viewModel`有`null`值，数据绑定也会给出`null`值判断（`viewModel.title`子表达式会给出`"null"`），保证应用不会因为这个原因而崩溃。

20.8.3 BindingAdapter

数据绑定默认会把绑定表达式解读为属性方法调用。所以，以下代码会被翻译为`setText(String)`方法调用。

```
android:text="@{`File name: ` + viewModel.title ?? `No file`}"
```

然而，这还不算什么。有时候，你可能会想给某些特别属性赋予一些定制行为。一般的做法是写一个`BindingAdapter`：

```
public class BeatBoxBindingAdapter {
    @BindingAdapter("app:soundName")
    public static void bindAssetSound(Button button, String assetFileName) {
        ...
    }
}
```

很简单，在项目的任何类里创建一个静态类，再使用`@BindingAdapter`，并传入想绑定作为参数的属性名即可。（是的，这就可以了。）数据绑定只要想用那个属性，它就会调用这个静态方法。

对于标准库里的组件，很可能你也想用数据绑定做点什么。实际上，有一些常见的操作已经定义了绑定adapter。例如，`TextViewBindingAdapter`就为`TextView`提供了一些特别的属性操作。你可以看看它们的源码。所以，若想自己写解决方案，不妨先按Command+Shift+O（Ctrl+Shift+O）搜一搜，打开其关联的绑定adapter看看，也许已经有你想要的了。

20.9　深入学习：为何使用 assets

事实上，BeatBox应用也可以使用Android资源处理。资源可以存储声音文件。比如在res/raw目录下保存79_long_scream.wav文件后，就可以使用像`R.raw.79_long_scream`这样的ID取到它。声音文件存储为资源后，就可以像使用其他资源那样使用它们了。例如，可以根据设备的不同方位、语言以及系统版本调用不同的声音资源。

那么为何还要选assets呢？这是因为，BeatBox应用要用到很多声效，总共大约20多个声音文件。可以想象，如果使用Android资源系统一个个去处理，效率会非常低。要是这些文件能全放在一个目录下管理就好了，可惜资源系统不允许这么做。

资源系统做不到的，就是assets大显身手的地方。assets可以看作随应用打包的微型文件系统，支持任意层次的文件目录结构。因为这个优点，assets常用来加载大量图片和声音资源，如游戏这样的应用。

20.10　深入学习：什么是 non-assets

`AssetManager`类还有像`openNonAssetFd(...)`这样的方法。想不通吧，assets专属类为什么要关心non-assets？我们也许会答道："你也没必要关心这事！"算了，你就当从未听说过它好了。

就我们所知，还真找不到使用它的理由，所以不学习它的理由很充分。

不过，既然你花钱买了这本书，下面就简单介绍一下，博君一笑好了。

前面说过，Android有assets和resources两大资源系统。resources系统有良好的检索机制，但无法处理大资源。这些大资源，如声音文件和图像文件，通常会保存在assets系统里。在后台，Android就是使用`openNonAsset`方法来打开它们的。不过，这样的方法有不少没对用户开放。

现在了解了吧！有机会用到它吗？永远没有。

第 21 章 音频播放与单元测试

MVVM架构极大方便了一项关键编程工作：单元测试。这也是其受追捧的另一个原因。单元测试是指编写小程序去验证应用各个单元的独立行为。BeatBox应用的单元是Java类。单元测试就是测试这些类。

BeatBox的音频资源文件已准备就绪，我们在本章学习如何播放这些.wav音频文件。在创建音频播放功能并整合的过程中，还会对SoundViewModel的功能整合做单元测试。

Android的大部分音频API都比较低级，不易掌握。不过没关系，对于BeatBox应用，可以使用SoundPool这个定制版实用工具。SoundPool能加载一批声音资源到内存中，并能控制同时播放的音频文件的个数。所以，就算用户一时兴奋，狂按各个按钮播放音频，也不用担心会搞坏应用或让手机掉电。

准备好了吗？开始吧。

21.1 创建 SoundPool

首先实现音频播放功能，这需要创建一个SoundPool对象，如代码清单21-1所示。

代码清单21-1 创建SoundPool对象（BeatBox.java）

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";
    private static final int MAX_SOUNDS = 5;

    private AssetManager mAssets;
    private List<Sound> mSounds = new ArrayList<>();
    private SoundPool mSoundPool;

    public BeatBox(Context context) {
        mAssets = context.getAssets();
        // This old constructor is deprecated, but we need it for compatibility.
        mSoundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0);
        loadSounds();
    }
    ...
}
```

Lollipop引入了新的SoundPool创建方式：使用SoundPool.Builder。不过，为了兼容API 19这一最低级别，还是要用SoundPool(int, int, int)这个老构造方法。

第一个参数指定同时播放多少个音频。这里指定了5个。已经播放了5个音频时，如果尝试再播第6个，SoundPool会停止播放原来的音频。

第二个参数确定音频流类型。Android有很多不同的音频流，它们都有各自独立的音量控制选项。这就是调低音乐音量，闹钟音量却不受影响的原因。打开开发者文档，先找到AudioManager类中以AUDIO打头的常量，再看看其他控制选项。STREAM_MUSIC是音乐和游戏常用的音量控制常量。

最后一个参数指定采样率转换品质。参考文档说这个参数不起作用，所以这里传入0。

21.2　加载音频文件

接下来使用SoundPool加载音频文件。相比其他音频播放方法，SoundPool还有个快速响应的优势：指令刚一发出，它就立即开始播放，一点都不拖沓。

不过反应快也要付出代价，那就是在播放前必须预先加载音频。SoundPool加载的音频文件都有自己的Integer型ID。如代码清单21-2所示，在Sound类中添加mSoundId实例变量，并添加相应的getter方法和setter方法管理这些ID。

代码清单21-2　添加mSoundId实例变量（Sound.java）

```
public class Sound {
    private String mAssetPath;
    private String mName;
    private Integer mSoundId;
    ...
    public String getName() {
        return mName;
    }

    public Integer getSoundId() {
        return mSoundId;
    }

    public void setSoundId(Integer soundId) {
        mSoundId = soundId;
    }
}
```

注意，mSoundId用了Integer类型而不是int。这样，在Sound的mSoundId没有值时，可以设置其为null值。

现在处理音频加载。在BeatBox中添加load(Sound)方法载入音频，如代码清单21-3所示。

代码清单21-3　加载音频（BeatBox.java）

```
    private void loadSounds() {
```

```
        ...
    }

    private void load(Sound sound) throws IOException {
        AssetFileDescriptor afd = mAssets.openFd(sound.getAssetPath());
        int soundId = mSoundPool.load(afd, 1);
        sound.setSoundId(soundId);
    }

    public list<Sound> getSounds() {
        return mSounds;
    }
}
```


调用mSoundPool.load(AssetFileDescriptor, int)方法可以把文件载入SoundPool待播。为了方便管理、重播或卸载音频文件，mSoundPool.load(...)方法会返回一个int型ID。这实际就是存储在mSoundId中的ID。调用openFd(String)方法有可能抛出IOException，load(Sound)方法也是如此。

现在，在BeatBox.loadSounds()方法中，调用load(Sound)方法载入全部音频文件，如代码清单21-4所示。

代码清单21-4　载入全部音频文件（BeatBox.java）

```
private void loadSounds() {
    ...
    for (String filename : soundNames) {
        try {
            String assetPath = SOUNDS_FOLDER + "/" + filename;
            Sound sound = new Sound(assetPath);
            load(sound);
            mSounds.add(sound);
        } catch (IOException ioe) {
            Log.e(TAG, "Could not load sound " + filename, ioe);
        }
    }
}
```

运行应用确认音频都已正确加载。否则，会看到LogCat中的红色异常日志。

21.3　播放音频

最后一步是播放音频。在BeatBox中添加play(Sound)方法。

代码清单21-5　播放音频（BeatBox.java）

```
public BeatBox(Context context) {
    mAssets = context.getAssets();
    // This old constructor is deprecated but needed for compatibility
    mSoundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0);
    loadSounds();
```

```
    }

    public void play(Sound sound) {
        Integer soundId = sound.getSoundId();
        if (soundId == null) {
            return;
        }
        mSoundPool.play(soundId, 1.0f, 1.0f, 1, 0, 1.0f);
    }

    private void loadSounds() {
```

播放前，要检查并确保soundId不是null值。Sound加载失败会出现null值的情况。

检查通过后，就可以调用SoundPool.play(int, float, float, int, int, float)方法播放音频了。这些参数依次是：音频ID、左音量、右音量、优先级（无效）、是否循环以及播放速率。我们需要最大音量和常速播放，所以传入值1.0。是否循环参数传入0，代表不循环。（如果想无限循环，可以传入-1。相信这会异常讨人厌。）

现在，可以把音频播放功能整合进SoundViewModel了。不过，我们打算先做单元测试再整合。具体做法是这样：先写个肯定会失败的单元测试，然后整合，让单元测试成功通过。

21.4　添加测试依赖

要编写测试代码，首先需要添加两个测试工具：Mockito和Hamcrest。Mockito是一个方便创建虚拟对象的Java框架。有了虚拟对象，就可以单独测试SoundViewModel，不用担心会因代码关联关系测到其他对象。

Hamcrest是个规则匹配器工具库。匹配器可以方便地在代码里模拟匹配条件。如果不能按预期匹配条件定义，测试就通不过。这可以验证代码是否按预期工作。

有这两个依赖库就可以做单元测试了，现在就来添加。右键单击app模块，选择Open Module Settings菜单项。选择弹出界面里的Dependencies选项页，然后点击+按钮弹出选择依赖库窗口，输入mockito后搜索，如图21-1所示。

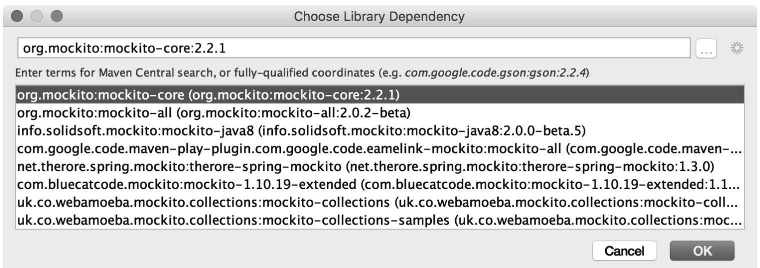


图21-1　导入Mockito

选择org.mockito:mockito-core依赖库，点击OK按钮完成添加。重复上述步骤，搜索

hamcrest-junit并选择org.hamcrest:hamcrest-junit完成对Hamcrest依赖库的添加。

完成之后，就可以在依赖清单里看到了。注意，这两个依赖项右边都有下拉箭头。点击它，可选择不同的依赖项作用范围。由于只允许指定整合测试范围，因此需要手动修改build.gradle文件。

打开build.gradle文件，把依赖项作用范围从`compile`改为`testCompile`，如代码清单21-6所示。

代码清单21-6　修改依赖项作用范围（app/build.gradle）

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:24.2.0'
    testCompile 'junit:junit:4.l2'
    compile 'com.android.support:recyclerview-v7:24.2.0'
    compile testCompile 'org.mockito:mockito-core:2.2.1'
    compile testCompile 'org.hamcrest:hamcrest-junit:2.0.0.0'
}
```

`testCompile`作用范围表示，这两个依赖项只需包括在应用的测试编译里。这样就能避免在APK包里捎带上无用代码库。

21.5　创建测试类

写单元测试最方便的方式是使用测试框架。使用测试框架可以集中编写和运行测试案例，并支持在Android Studio里看到测试结果。

JUnit是最常用的Android单元测试框架，能和Android Studio无缝整合。要用它测试，首先要创建一个用作JUnit测试的测试类。打开SoundViewModel.java文件，使用Command+Shift+T（Ctrl+Shift+T）组合键。Android Studio会尝试寻找这个类关联的测试类。如果找不到，它就会提示新建，如图21-2所示。

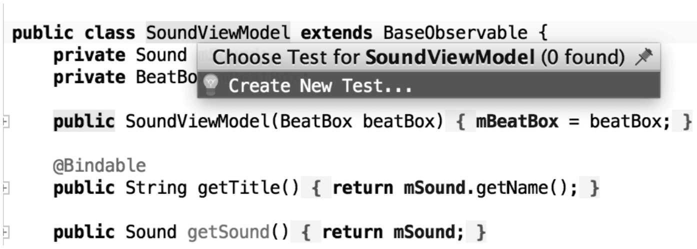


图21-2　尝试打开测试类

选择Create New Test...创建一个新测试类。测试库选择JUnit4，勾选setUp/@Before，其他保持默认设置，如图21-3所示。

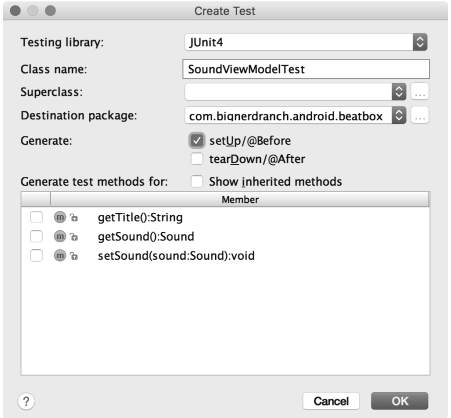


图21-3　创建测试类

点击OK按钮，进入下一个对话框。

最后一步是选择创建哪种测试类，或者说选择哪个测试目录存放测试类（androidTest和test）。在androidTest目录下的都是整合测试类。整合测试可以运行在设备或虚拟设备上。这样做有优点：可以在运行时动态测试应用行为。但也有缺点：需要编译打包为APK在设备上运行，浪费资源。（详见21.12节。）

在test目录下的是单元测试类。单元测试运行在本地开发机上，可以脱离Android运行时环境，因此速度会快很多。

单元测试的规模最小：测试单个类。所以，单元测试不需要运行整个应用或支持设备，可以不影响手头工作，快速反复地执行。考虑到这个因素，我们选择test目录存放测试类（如图21-4），最后点击OK按钮完成。

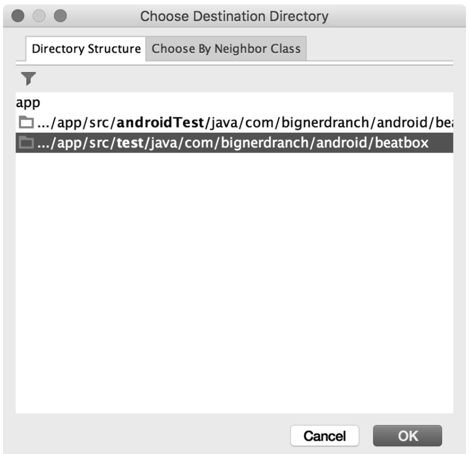


图21-4　选择test目录

21.6　实现测试类

现在来实现SoundViewModel测试类。测试框架创建的模板类只有一个setUp()方法，如代码清单21-7所示。

代码清单21-7　空测试类（SoundViewModelTest.java）

```
public class SoundViewModelTest {
    @Before
    public void setUp() throws Exception {

    }
}
```

（这个测试类位于app模块下的test目录下。）

和大多数对象一样，测试类也需要创建对象实例以及它依赖的其他对象。为了避免为每一个测试类写重复代码，JUnit提供了@Before这个注解。以@Before注解的包含公共代码的方法会在所有测试之前运行一次。按照约定，所有单元测试类都要有以@Before注解的setUp()方法。

使用虚拟依赖项

在setUp()方法里，我们会创建一个SoundViewModel实例用来测试。这需要BeatBox实例，因为SoundViewModel需要BeatBox作为构造参数。

在实际应用中，当然会创建BeatBox实例。怎么创建？像下面这样：

```
SoundViewModel viewModel = new SoundViewModel(new BeatBox());
```

如果在单元测试中也这样做，会有一个问题：如果BeatBox出了问题，很明显，用到它的测试代码都会出错。

解决办法很简单：使用虚拟BeatBox。虚拟对象会继承BeatBox，有同样的方法，但这些方法啥事都不干。这样，依赖BeatBox的SoundViewModel测试就不会有问题了。

要用Mockito创建虚拟对象，需要传入要虚拟的类，调用mock(Class)静态方法。创建一个虚拟BeatBox对象并存入mBeatBox变量，如代码清单21-8所示。

代码清单21-8　创建虚拟BeatBox对象（SoundViewModelTest.java）

```
public class SoundViewModelTest {
    private BeatBox mBeatBox;

    @Before
    public void setUp() throws Exception {
        mBeatBox = mock(BeatBox.class);
    }
}
```

使用mock(Class)方法需要导入支持包，这和其他类引用没什么两样。mock(Class)方法会自动创建一个虚拟版本的BeatBox。这确实很方便。

有了虚拟依赖对象，现在来完成SoundViewModel测试类。创建一个SoundViewModel和一个Sound备用（Sound是简单的数据对象，不容易出问题，这里就虚拟它了），如代码清单21-9所示。

代码清单21-9 创建SoundViewModel测试对象（SoundViewModelTest.java）

```
public class SoundViewModelTest {
    private BeatBox mBeatBox;
    private Sound mSound;
    private SoundViewModel mSubject;

    @Before
    public void setUp() throws Exception {
        mBeatBox = mock(BeatBox.class);
        mSound = new Sound("assetPath");
        mSubject = new SoundViewModel(mBeatBox);
        mSubject.setSound(mSound);
    }
}
```

注意，在本书的其他地方，声明SoundViewModel类型变量时，命名一般是mSoundViewModel。这里，我们用了mSubject。这是一种习惯约定，这样做的原因有两点：

- 很清楚就知道，mSubject是要测试的对象（与其他对象区别开来）；
- 如果SoundViewModel里有任何方法要移到其他类，比如BeatBoxSoundViewModel，那么测试方法可以直接复制过去，省了mSoundViewModel到mBeatBoxSoundViewModel重命名的麻烦。

21.7 编写测试方法

setUp()支持方法完成了，现在可以写测试代码了。实际上，就是在测试类里写一个以@Test注解的测试方法。

如代码清单21-10所示，首先写一个方法，断定SoundViewModel里的getTitle()属性和Sound里的getName()属性是有关系的。

代码清单21-10 测试标题属性（SoundViewModelTest.java）

```
@Before
public void setUp() throws Exception {
    mBeatBox = mock(BeatBox.class);
    mSound = new Sound("assetPath");
    mSubject = new SoundViewModel(mBeatBox);
    mSubject.setSound(mSound);
}

@Test
public void exposesSoundNameAsTitle() {
    assertThat(mSubject.getTitle(), is(mSound.getName()));
}
```

注意，窗口里有两个方法会变红：`assertThat(...)`方法和`is(...)`方法。在`assertThat(...)`方法上使用Option+Return（Alt+Enter）组合键，然后选Static import method...，然后选hamcrest-core-1.3库里的`MatcherAssert.assertThat(...)`方法。以同样方式，选hamcrest-core-1.3库里的`Is.is`方法。

这个测试方法使用了Hamcrest匹配器的`is(...)`方法和JUnit的`assertThat(...)`方法。方法体里的代码很直白：断定测试对象获取标题方法和sound的获取文件名方法返回相同的值。如果不同，单元测试失败。

为了运行测试，右键点击app/java/com.bignerdranch.android.beatbox (test)，然后选择Run 'Tests in 'beatbox''。随后，一个结果窗口弹出，如图21-5所示。



图21-5　测试过关

测试结果窗口默认只会显示失败的测试。所以，你知道，测试通过了。

测试对象交互

刚才做了测试热身，现在处理关键任务：整合`SoundViewModel`和`BeatBox.play(Sound)`方法。实践中，通常的做法是，在写新方法之前，先写一个测试验证这个方法的预期结果。我们需要在`SoundViewModel`类里写`onButtonClicked()`方法去调用`BeatBox.play(Sound)`方法。如代码清单21-11所示，写一个测试方法调用`onButtonClicked()`方法。

代码清单21-11　测试`onButtonClicked()`方法（SoundViewModelTest.java）

```
    @Test
    public void exposesSoundNameAsTitle() {
        assertThat(mSubject.getTitle(), is(mSound.getName()));
    }

    @Test
    public void callsBeatBoxPlayOnButtonClicked() {
        mSubject.onButtonClicked();
    }
}
```

注意，这个方法还没写，所以它是红色的。如代码清单21-12所示，将光标移到它身上，按Option+Return（Alt+Enter）组合键，然后选Create method 'onButtonClicked'创建这个方法。

代码清单21-12　创建onButtonClicked()方法（SoundViewModel.java）

```
    public void setSound(Sound sound) {
        mSound = sound;
        notifyChange();
    }

    public void onButtonClicked() {

    }
}
```

先不管这个空方法，按Command+Shift+T（Ctrl+Shift+T）组合键，回到SoundViewModelTest测试类。

单元测试方法会调用这个方法，而且，也应验证这个方法的实际作用：调用BeatBox.play(Sound)方法。这种繁琐的事就交给Mockito吧！对于每次调用，所有的Mockito虚拟对象都能自我跟踪管理哪些方法调用了，以及都传入了哪些参数。

如代码清单21-13所示，调用verify(Object)方法，确认onButtonClicked()方法调用了BeatBox.play(Sound)方法。

代码清单21-13　验证BeatBox.play(Sound)方法是否调用（SoundViewModelTest.java）

```
        assertThat(mSubject.getTitle(), is(mSound.getName()));
    }

    @Test
    public void callsBeatBoxPlayOnButtonClicked() {
        mSubject.onButtonClicked();

        verify(mBeatBox).play(mSound);
    }
}
```

类似于前面的AlertDialog.Builder类，verify(Object)使用了流接口，分开写就像这样：

```
verify(mBeatBox);
mBeatBox.play(mSound);
```

调用verify(mBeatBox)方法就是说："我要验证mBeatBox对象的某个方法是否调用了。"紧跟的mBeatBox.play(mSound)方法是说："验证这个方法是这样调用的。"所以，合起来就是说："验证以mSound作为参数，调用了mBeatBox对象的play(...)方法。"

SoundViewModel.onButtonClicked()是个空方法，所以，什么也没发生。这意味着测试应该会失败。因为是先写测试，所以这不是问题。如果测试不失败，那说明啥也没测到。

运行测试看结果。（可以按上述步骤运行测试，也可以使用Command+R（Ctrl+R）快捷键重复上一次Run命令。）运行结果如图21-6所示。

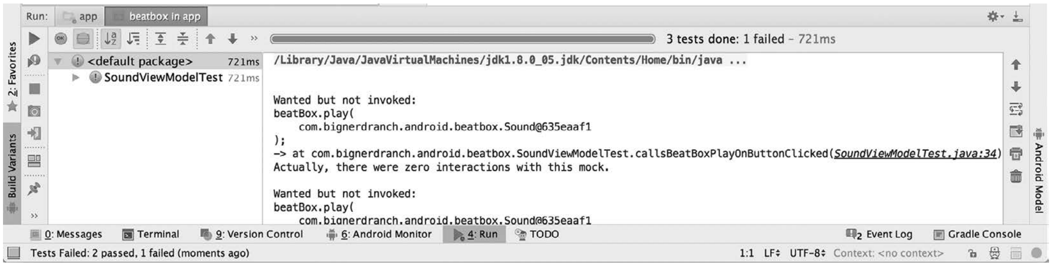


图21-6 测试失败输出

测试结果表明，测试方法要调用mBeatBox.play(mSound)，但没成功：

```
Wanted but not invoked:
beatBox.play(
    com.bignerdranch.android.beatbox.Sound@64cd705f
);
-> at ….callsBeatBoxPlayOnButtonClicked(SoundViewModelTest.java:28)
Actually, there were zero interactions with this mock.
```

这实质上是说，和assertThat(...)方法一样，verify(Object)做出某个断定，但断定无效，测试失败并给出问题原因日志。

如代码清单21-14所示，实现onButtonClicked()方法，让测试符合预期。

代码清单21-14 实现onButtonClicked()方法（SoundViewModel.java）

```
    public void setSound(Sound sound) {
        mSound = sound;
        notifyChange();
    }

    public void onButtonClicked() {
        mBeatBox.play(mSound);
    }
}
```

再次运行测试。如图21-7所示，这次一路绿灯，测试顺利通过。

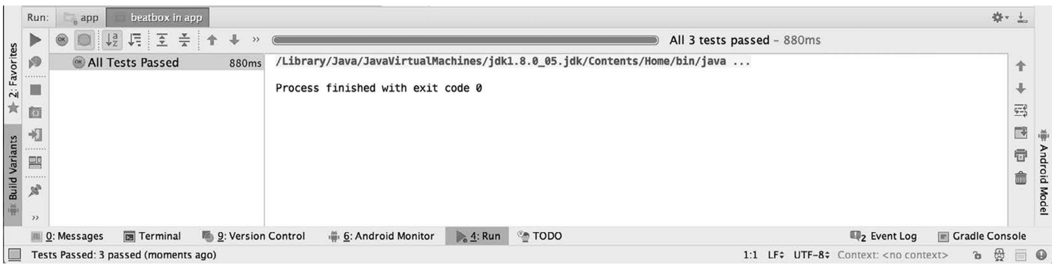


图21-7 测试全部过关

21.8　数据绑定回调

按钮要响应事件还差最后一步：关联按钮对象和`onButtonClicked()`方法。

和前面使用数据绑定关联数据和UI一样，你也可以使用lambda表达式，让数据绑定帮忙关联按钮和点击监听器（如果忘了，请参见20.8.1节）。

如代码清单21-15所示，在布局文件里，添加数据绑定lambda表达式，让按钮对象和`onButtonClicked()`方法关联起来。

代码清单21-15　关联按钮（list_item_sound.xml）

```
<Button
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:onClick="@{() -> viewModel.onButtonClicked()}"
    android:text="@{viewModel.title}"
    tools:text="Sound name"/>
```

现在，如果运行应用，按钮就能播放声音。然而，如果你尝试使用绿色的运行按钮，测试又运行了。这是因为右键点击运行测试修改了运行配置。这个配置决定点击绿色按钮之后，Android Studio该运行什么。

如图21-8所示，为了运行BeatBox应用，点击Run按钮旁边的配置选择器，切换至app运行配置。

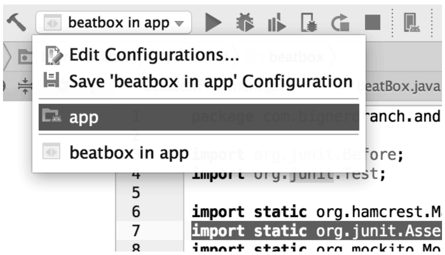


图21-8　切换运行配置

运行BeatBox应用，点击按钮。你会听到各种吓人的喊叫声。不要害怕，前面说过，这个应用就是用来吓人的。

21.9　释放音频

BeatBox应用可用了，但别忘了做善后工作。音频播放完毕，应调用`SoundPool.release()`方法释放`SoundPool`。如代码清单21-16所示，添加`BeatBox.release()`清理方法。

代码清单21-16　释放SoundPool（BeatBox.java）

```
public class BeatBox {
```

```
    ...
    public void play(Sound sound) {
        ...
    }

    public void release() {
        mSoundPool.release();
    }
    ...
}
```

同样，在BeatBoxFragment中，也完成释放，如代码清单21-17所示。

代码清单21-17　释放BeatBox（BeatBoxFragment.java）

```
public class BeatBoxFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mBeatBox.release();
    }
    ...
}
```

再次运行应用，确认添加release()方法后，应用工作正常。尝试播放一长段声音，同时旋转设备或点按后退键，声音播放应该会停止。

21.10　设备旋转和对象保存

处理完资源释放，再来看一个经典的设备旋转问题。播放69_ohm-loko这个音频，然后旋转设备，声音戛然而止。（如果没有，请确认已实现onDestroy()方法，并重新编译和运行应用。）

下面分析一下这个问题：设备旋转时，BeatBoxActivity随即被销毁。与此同时，FragmentManager也会销毁BeatBoxFragment。在销毁过程中，它会逐一调用BeatBoxFragment的生命周期方法onPause()、onStop()和onDestroy()。在BeatBoxFragment.onDestroy()方法中，BeatBox.release()方法会被调用。这会释放SoundPool，音频播放自然也就停止了。

前面，我们遇到过Activity和Fragment因设备旋转而被销毁的问题。当时使用onSave-InstanceState(Bundle)方法解决了问题。然而，老办法在这里行不通，因为需要首先保存数据，然后再使用Bundle中的Parcelable恢复数据。

类似于Serializable，Parcelable是一个把对象以字节流的方式保存的API。对于可保存对象，可以让它实现Parcelable接口。在Java世界，要保存对象，要么将其放入Bundle中，要

么实现Serializable接口或者Parcelable接口。无论采用哪种方式,对象首先要是可保存对象。

怎么理解可保存呢？举个例子你就明白了。你和朋友在看电视节目。什么频道、音量大小，甚至是电视型号，你都可以记下来。如此一来，就算发生火灾或停电这样的事情，等一切恢复正常，看看记下的信息，你依然能接着看原来的电视节目。

显然，当前所看的电视节目的配置是可保存的，而观看节目的那段时间却无法保存：一旦发生火灾或停电，其间那段时光就流逝掉了。就算恢复观看，流逝掉的那段时光再也找不回来了。所以，什么能保存，什么不能保存，再清楚不过了。

BeatBox的某些部分可以保存，例如，Sound类中的一切都可以保存；而SoundPool就无法保存了。虽然可以新建包含同样音频文件的SoundPool，甚至能从音频播放中断处继续，你还是会体验到被打断的滋味。这是改变不了的事实。所以说，SoundPool是无法保存的。

不可保存性有向外传递的倾向。如果一个对象重度依赖另一个不可保存的对象，那么这个对象很可能也无法保存。BeatBox和SoundPool就是这样的一对。SoundPool要依靠BeatBox播放音频。基于这个事实，可以证明BeatBox也是无法保存的。（抱歉。）

普通的savedInstanceState机制只适用于可保存的对象数据，但BeatBox不可保存。在Activity创建和销毁时，BeatBox实例需要持续可用。

这个难题该怎么解决呢？

21.10.1　保留 fragment

告诉你一个好消息：为了应对设备配置变化，fragment有一个特殊方法可确保BeatBox实例不被销毁，这个方法就是retainInstance。覆盖BeatBoxFragment.onCreate(...)方法并设置fragment的属性值，如代码清单21-18所示。

代码清单21-18　调用setRetainInstance(true)（BeatBoxFragment.java）

```
public static BeatBoxFragment newInstance() {
    return new BeatBoxFragment();
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);

    mBeatBox = new BeatBox(getActivity());
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
```

fragment的retainInstance属性值默认为false，这表明其不会被保留。因此，设备旋转时fragment会随托管activity一起被销毁并重建。调用setRetainInstance(true)方法可保留

fragment。已保留的fragment不会随activity一起被销毁。相反，它会一直保留，并在需要时原封不动地转给新的activity。

对于已保留的fragment实例，其全部实例变量（如`mBeatBox`）的值也会保持不变，因此可放心继续使用。

运行BeatBox应用。播放69_ohm-loko声音文件，然后旋转设备，确认音频播放不受影响。

21.10.2 设备旋转和已保留的 fragment

解决了问题之后，我们来看看保留fragment的工作原理。fragment之所以能保留，是因为这样一个事实：可以销毁和重建fragment的视图，但fragment自身可以不被销毁。

设备配置发生改变时，`FragmentManager`首先销毁队列中fragment的视图。在设备配置改变时，总是销毁与重建fragment与activity的视图，这都是基于同样的理由：新的配置可能需要新的资源来匹配；当有更合适的资源可用时，则应重建视图。

紧接着，`FragmentManager`检查每个fragment的`retainInstance`属性值。如果属性值为`false`（初始默认值），`FragmentManager`会立即销毁该fragment实例。随后，为了适应新的设备配置，新activity的新`FragmentManager`会创建一个新的fragment及其视图，如图21-9所示。

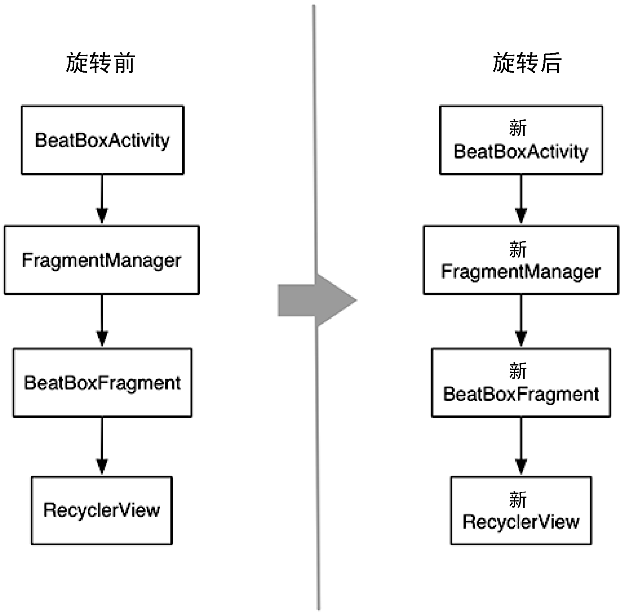


图21-9 设备旋转前后（UI fragment默认不保留）

如果属性值为`true`，则该fragment的视图立即被销毁，但fragment本身不会被销毁。为了适应新的设备配置，新activity创建后，新`FragmentManager`会找到已保留的fragment，并重新创建它的视图，如图21-10所示。

虽然已保留的fragment没有被销毁，但它已脱离消亡中的activity并处于保留状态。尽管此时的fragment还在，但已没有任何activity托管它，如图21-11所示。

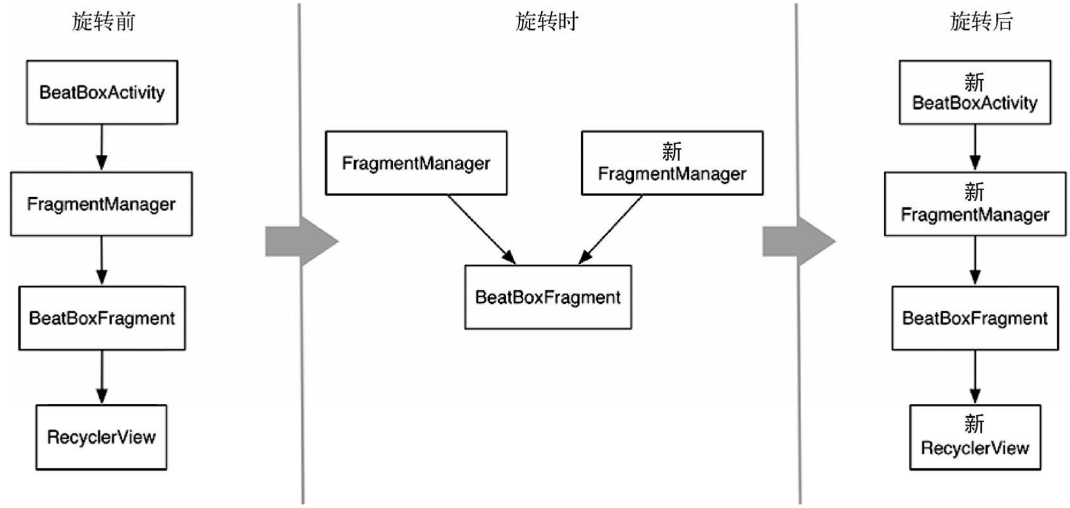


图21-10　设备旋转与已保留的UI fragment

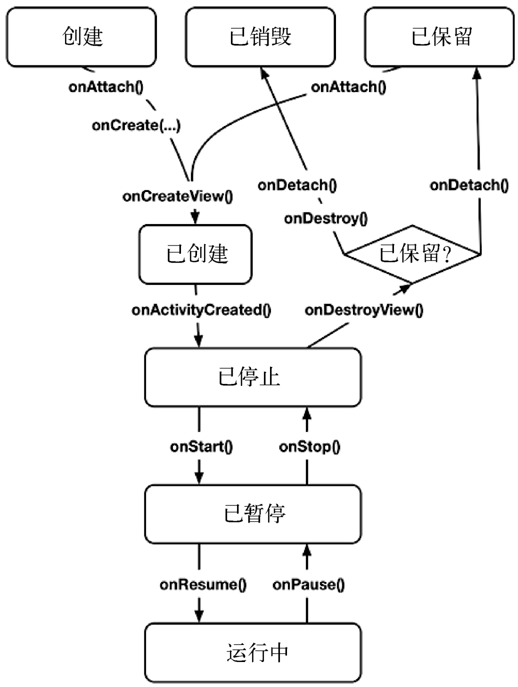


图21-11　fragment的生命周期

必须同时满足以下两个条件，fragment才能进入保留状态：

- 已调用了fragment的`setRetainInstance(true)`方法；
- 因设备配置改变（通常为设备旋转），托管activity正在被销毁。

fragment只能保留非常短的时间，即从fragment脱离旧activity到重新附加给快速新建的activity之间的一段时间。

21.11　深入学习：是否保留 fragment

保留fragment可以说是Android的一处巧妙设计，不是吗？没错！这似乎解决了因设备旋转而销毁activity和fragment所导致的全部问题。现在，如果设备配置有变，可以创建全新视图获取最合适的资源，也可以轻松保留原有数据及对象。

你可能会疑惑：为什么不保留所有fragment？为什么fragment的`retainInstance`默认属性值不是`true`？这是因为，除非万不得已，最好不要使用这种机制。下面给出理由。

首先，相比非保留fragment，已保留fragment用起来更复杂。一旦出现问题，问题排查非常耗时。既然它会让程序变得复杂，能不用就不用吧。

其次，fragment在使用保存实例状态的方式处理设备旋转时，也能够应对所有生命周期场景；但保留的fragment只能应付activity因设备旋转而被销毁的情况。如果activity是因系统回收内存而被销毁，则所有保留的fragment也会随之被销毁，数据也就跟着丢失了。

21.12　深入学习：Espresso 与整合测试

在测试`SoundViewModel`时，我们创建了`SoundViewModelTest`单元测试类。实际上，我们也可以选择创建整合测试。那么，什么是整合测试？

在单元测试里，受测对象都是单个类。在整合测试里，受测对象是整个应用。通常测试要覆盖每个页面。例如，我们会测试`BeatBoxActivity`界面出现后，第一个按钮显示的文件名是不是来自sample_sounds里的第一个文件：65_cjipie。

相比应用按设想实现，只有当应用按设想运行时，整合测试才算通过。修改某个按钮ID的名字并不会影响应用的功能。但是，如果你写了这样一个整合测试用例，“调用`findViewById(R.id.button)`方法，确认找到的按钮上的文字显示正确”，那么显然这个测试通不过。所以，整合测试应该用UI测试框架工具来写，而不是使用像`findViewById(int)`这样的标准库工具。这样可以很容易写出类似这样的用例：“确保屏幕上有个按钮，上面显示我设想的文字。”

Espresso是Google开发的一个UI测试框架，可用来测试Android应用。在app/build.gradle文件中，添加com.android.support.test.espresso:espresso-core依赖项，作用范围改为`androidTestCompile`，就可以引入它。

引入Espresso之后，就可以用它来测试某个activity的行为。例如，如果想断定屏幕上某个视图显示了第一个sample_sounds受测文件的文件名，就可以编写如下的测试用例：

```
@RunWith(AndroidJUnit4.class)
public class BeatBoxActivityTest {
    @Rule
    public ActivityTestRule<BeatBoxActivity> mActivityRule =
            new ActivityTestRule<>(BeatBoxActivity.class);
    @Test
    public void showsFirstFileName() {
        onView(withText("65_cjipie"))
                .check(matches(anything()));
    }
}
```

现在来解读一下样例代码。首先看其中的注解。`@RunWith(AndroidJUnit4.class)`表明，这是一个Android工具测试，需要activity和其他Android运行时环境支持。之后，`mActivityRule`上的`@Rule`注解告诉JUnit，运行测试前，要启动一个`BeatBoxActivity`实例。

准备工作做完，接下来就可以在测试方法里对`BeatBoxActivity`做断定测试了。在`showsFirstFileName()`方法里，`onView(withText("65_cjipie"))`这行代码会找到显示“65_cjipie”的视图，然后对其执行测试。`check(matches(anything()))`用来判定有这样的视图。如果没有，则测试失败。相较于JUnit的`assertThat(...)`断言方法，`check(...)`方法是Espresso版的断言方法。

有时，你可能还想点击某个视图，然后使用断言验证点击结果。可以让Espresso点击这个视图，或者使用下面这样的代码交互：

```
onView(withText("65_cjipie"))
        .perform(click());
```

与视图交互时，Espresso会等待应用闲置再执行下一个测试。Espresso有一套探测UI是否已更新完毕的方法。如果需要，也可使用`IdlingResource`的一个子类告诉Espresso：多等一会儿，应用还在忙。

有关如何使用Espresso做UI测试的更详细的信息，请阅读Espresso的文档（google.github.io/android-testing-support-library/docs/espresso）。

单元测试和整合测试用处各异。单元测试简单快速，多用用就会形成习惯，所以能让大多数人接受并喜欢。整合测试需要花很多时间，不适合做经常性的测试。然而，不管怎样，这两类测试都很重要，各自能从不同视角检验应用。所以，只要有条件，二者都不能少。

21.13　深入学习：虚拟对象与测试

相比单元测试，虚拟对象在整合测试中扮演了更为不寻常的角色。虚拟对象假扮成其他不相干的组件，其作用就是隔离受测对象。单元测试的受测对象是单个类；每个类都有自己不同的依赖关系，所以，每个受测类也有一套不同于其他类的虚拟对象。既然都是些不同的虚拟对象，那么它们各自的具体行为怎么样，怎么实现，一点也不重要。所以，对于单元测试来说，一些虚拟化框架，比如能快速创建虚拟对象的Mockito，就非常有用了。

再来看整合测试。在整合测试场景中，虚拟对象显然不能用来隔离应用，相反，我们用它把应用和可能的外部交互对象隔离开来，如提供web service假数据和假反馈。如果是在BeatBox应用里，你很可能就要提供虚拟`SoundPool`，让它告诉你某个声音文件何时播放。显然，相比常见的行为虚拟，这种虚拟太重了，而且还要在很多整合测试里共享。这真不如手动写假对象。所以，做整合测试时，最好避免使用像Mockito这样的自动虚拟测试框架。

21.14 挑战练习：播放进度控制

让用户快速多听一些声音，请给BeatBox应用添加播放进度控制功能。完成后的界面如图21-12所示。提示：在`BeatBoxFragment`中，使用`SeekBar`组件（developer.android.com/reference/android/widget/SeekBar.html）控制`SoundPool`的`play(int, float, float, int, int, float)`方法的播放速率参数值。



图21-12 带播放进度控制的BeatBox应用

文件。

第 22 章

样式与主题

既然BeatBox应用的音效能吓退人，它的用户界面也应透出一定的威慑力。

当前，BeatBox应用依然还是一副Android千年不变的老面孔。按钮普通，配色灰暗。整个应用看上去毫不起眼，没有品牌特色。

不过我们有技术，使用样式和主题，就能定制出漂亮的用户界面。

定制界面的最终效果如图22-1所示。与之前相比，新界面更加美观、惹眼，独具风格。



图22-1　应用了主题的BeatBox

22.1　颜色资源

首先，我们来定义本章要用到的颜色资源。参照代码清单22-1，在res/values中编辑colors.xml文件。

代码清单22-1　定义几种颜色（res/values/colors.xml）

```
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="red">#F44336</color>
    <color name="dark_red">#C3352B</color>
    <color name="gray">#607D8B</color>
    <color name="soothing_blue">#0083BF</color>
    <color name="dark_blue">#005A8A</color>
</resources>
```


使用颜色资源，可以方便地在一处定义各种颜色值，然后在整个应用里引用。

22.2　样式

现在，我们来给按钮添加样式。样式是能够应用于视图组件的一套属性。

打开res/values/styles.xml样式文件，添加BeatBoxButton新样式，如代码清单22-2所示。（创建BeatBox项目时，向导会创建默认的styles.xml文件。如果没有，请自行创建。）

代码清单22-2　添加样式（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="BeatBoxButton">
        <item name="android:background">@color/dark_blue</item>
    </style>

</resources>
```

新建样式名叫BeatBoxButton。该样式仅定义了android:background属性，属性值为深蓝色。样式可以为很多组件共用，更新修改属性时，只修改公共样式定义就行了。

定义好样式，把它添加给各个按钮，如代码清单22-3所示。

代码清单22-3　使用样式（res/layout/list_item_sound.xml）

```
<Button
    style="@style/BeatBoxButton"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:onClick="@{() -> viewModel.onButtonClicked()}"
    android:text="@{viewModel.title}"
    tools:text="Sound name"/>
```

运行BeatBox应用。可以看到，所有按钮的背景都是深蓝色了，如图22-2所示。



图22-2　添加了样式的按钮

如果需要，可以创建带多套属性的样式在应用里复用。这真是方便。

样式继承

样式支持继承。一个样式能继承并覆盖其他样式的属性。

创建一个名叫BeatBoxButton.Strong的新样式。除了继承BeatBoxButton样式的按钮背景属性，再添加自己的android:textStyle属性，用粗体显示按钮文字，如代码清单22-4所示。

代码清单22-4　继承样式（res/layout/styles.xml）

```
<style name="BeatBoxButton">
    <item name="android:background">@color/dark_blue</item>
</style>

<style name="BeatBoxButton.Strong">
    <item name="android:textStyle">bold</item>
</style>
```

（当然，可以直接对BeatBoxButton样式添加这个android:textStyle属性。创建BeatBox-Button.Strong样式只是为了演示样式继承。）

新样式的命名有点特别。BeatBoxButton.Strong的命名表明，这个新样式继承了BeatBoxButton样式的属性。

除了通过命名表示样式继承关系，也可以采用指定父样式的方式：

```
<style name="BeatBoxButton">
    <item name="android:background">@color/dark_blue</item>
</style>

<style name="StrongBeatBoxButton" parent="@style/BeatBoxButton">
    <item name="android:textStyle">bold</item>
</style>
```

虽然有新方式用于继承样式，BeatBox应用还是继续使用特殊命名方式。

更新list_item_sound.xml布局，用上新的粗体文字样式，如代码清单22-5所示。

代码清单22-5　使用粗体文字样式（res/layout/list_item_sound.xml）

```
<Button
    style="@style/BeatBoxButton.Strong"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:onClick="@{() -> viewModel.onButtonClicked()}"
    android:text="@{viewModel.title}"
    tools:text="Sound name"/>
```

运行应用，确认按钮文字已显示为粗体，如图22-3所示。



图22-3　使用了粗体文字样式的BeatBox

22.3　主题

样式很有用。在styles.xml公共文件中，可以为所有组件定义一套样式属性共用。可惜，定义公共样式属性虽方便，实际应用却很麻烦：需要逐个为所有组件添加它们要用到的样式。要是开

发一个复杂应用，涉及很多布局、无数按钮，仅仅添加样式就累死人了。

该是主题闪亮登场的时候了！主题可看作样式的进化加强版。同样是定义一套公共主题属性，样式属性需要逐个添加，而主题属性则会自动应用于整个应用。主题属性能引用颜色这样的外部资源，也能引用其他样式。使用主题，不用找到每个按钮，告诉它们要用哪个主题。一句话就搞定："所有按钮都使用这个样式。"

修改默认主题

创建BeatBox项目时，向导给了它默认主题。找到并打开AndroidManifest.xml文件，可以看到application标签下的theme属性，如代码清单22-6所示。

代码清单22-6　BeatBox的默认主题（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.beatbox" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        ...
    </application>

  </manifest>
```

theme属性指向的主题叫AppTheme。它也定义在styles.xml文件中。

可见，主题实际就是一种样式。但是主题指定的属性有别于样式。（稍后就会看到。）既然能在manifest文件中声明它，主题威力大增。同时解释了为什么主题可以自动应用于整个应用。

要查看AppTheme主题定义，只要按住Command键（Windows系统是Ctrl键），点击@style/AppTheme，Android Studio就会自动打开res/values/styles.xml文件，如代码清单22-7所示。

代码清单22-7　BeatBox的AppTheme（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        ...
    </style>

    <style name="BeatBoxButton">
        <item name="android:background">@color/dark_blue</item>
    </style>
    ...
</resources>
```

（本书编写时，在Android Studio中创建的项目都自带AppCompat主题。如果你的BeatBox项目没有，请参考第13章让项目使用AppCompat库。）

AppTheme现在继承Theme.AppCompat.Light.DarkActionBar的全部属性。如有需要，可以添加自己的属性值，或是覆盖父主题的某些属性值。

AppCompat库自带三大主题：

- Theme.AppCompat——深色主题
- Theme.AppCompat.Light——浅色主题
- Theme.AppCompat.Light.DarkActionBar——带深色工具栏的浅色主题

把AppTheme的父主题修改为Theme.AppCompat，如代码清单22-8所示。这样BeatBox项目就有了一个深色主题基板。

代码清单22-8　改用深色主题（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

    </style>
    ...
</resources>
```

运行BeatBox应用，查看新的深色主题，如图22-4所示。



图22-4　应用了深色主题的BeatBox

22.4　添加主题颜色

现在，基于AppTheme主题模板，我们来定制它的属性。

在styles.xml文件中，参照代码清单22-9修改现有三个属性。

代码清单22-9 自定义主题属性（res/values/styles.xml）

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="colorPrimary">@color/colorPrimary red</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark dark_red</item>
    <item name="colorAccent">@color/colorAccent gray</item>
</style>
```

虽然这三个主题属性看上去和前面的样式属性差不多，但它们的应用范围不一样。样式属性仅适用于单个组件，如前面用粗体显示按钮文字的`textStyle`。主题属性则适用所有使用同一主题的组件。例如，工具栏会以主题的`colorPrimary`属性设置自己的背景色。

使用这三个主题属性，应用界面大有改观。`colorPrimary`属性主要用于工具栏。由于应用名称是显示在工具栏上的，`colorPrimary`也可以称为应用品牌色。

`colorPrimaryDark`用于屏幕顶部的状态栏。从名字可以看出，它是深色版`colorPrimary`。注意，只有Lollipop以后的系统支持状态栏主题色。对于之前的系统，无论指定什么主题色，状态栏都是不变的黑底色。图22-5展示了这两种主题色的应用效果。

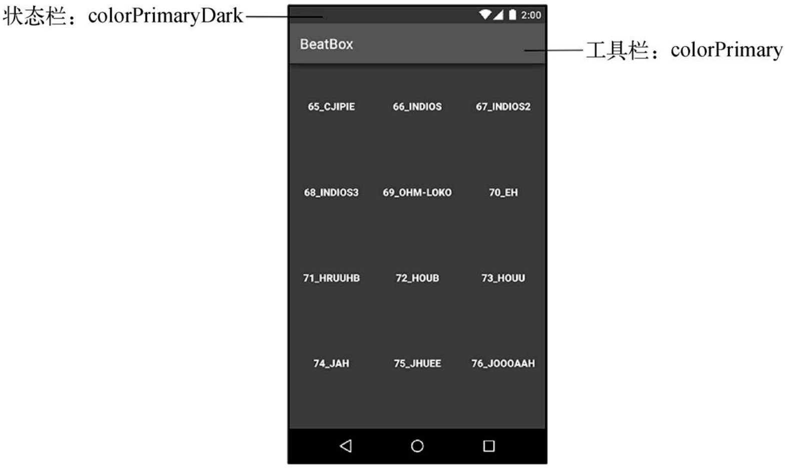


图22-5 带AppCompat颜色属性的BeatBox

最后，设置`colorAccent`为灰色。这个主题色应该和`colorPrimary`形成反差效果，主要用于给`EditText`这样的组件着色。

按钮组件不支持着色，所以`colorAccent`主题色在BeatBox项目中没有效果。不管有没有用，这里还是添加了`colorAccent`，因为最好能配套使用这三个颜色属性。现在，刚设置的主题色融合得很好（继承自父主题的默认`colorAccent`可能会和你指定的其他两种主题色冲突），算是打下了个好基础。

运行应用查看主题效果。现在，应用界面看起来应该和图22-5差不多。

22.5　覆盖主题属性

完成了主题配色，我们继续深入，看看都有哪些主题属性可以覆盖。给你提个醒，主题深究之路坎坷崎岖，可不那么好走。有哪些主题属性可用，哪些能覆盖，甚至是某些属性究竟有什么作用，研究诸如此类的问题时，没有官方参考文档还是小事，极有可能你就完全没了方向。对此，我们只有一个建议：阅读本书。

第一个任务是修改主题以更换BeatBox应用的背景色。当然，你可以打开res/layout/fragment_beat_box.xml文件，手工设置`RecyclerView`视图的`android:background`属性。如果还有其他fragment和activity要改，都照此处理。这简直是浪费：浪费你的时间，浪费应用资源。

主题已经设置了背景色，在此基础上再设置其他颜色，就是自己给自己找事。而且，在应用里到处复制使用背景属性设置代码也不利于后期维护。

主题探秘

要解决上述问题，应设法覆盖主题背景色属性。为了找出可覆盖属性的名字，先来看看这个属性在其`Theme.AppCompat`父主题里是怎么设置的。

你可能会想："不知道名字，我怎么知道该覆盖哪个属性呢？"确实是这样。所以，首先查看目标属性的名字，凭直觉挑一个，覆盖它，然后运行应用验证猜想。

你需要找出主题继承的源头。主题继承树有多深，谁也不知道，只能一层层向上找，一直找到目标为止。

打开styles.xml文件，按住Command键（Windows系统是Ctrl键）点击`Theme.AppCompat`，来看看继承有多深。

（如果无法直接在Android Studio里追溯主题属性，或是想在工具之外查找，可以在your-SDK-directory/platforms/android-24/data/res/values目录找到主题源码。）

Android开发工具更新频繁，本书编写时，Android Studio会定位到一个大文件的这行：

```
<style name="Theme.AppCompat" parent="Base.Theme.AppCompat" />
```

可知`Theme.AppCompat`主题属性继承自`Base.Theme.AppCompat`。有趣的是，`Theme.AppCompat`本身没有覆盖任何属性，仅仅指向了其父主题。

按住Command键再点击`Base.Theme.AppCompat`，Android Studio会提示，这个主题有资源修饰符，有多个版本可选。

选择values/values.xml版本，定位到如图22-6所示的主题定义。

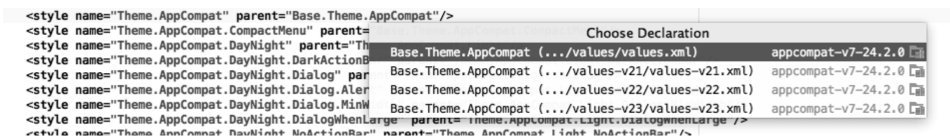


图22-6　选择父主题

（BeatBox支持19及以上API级别，所以这里选择了无修饰版本。如果选择v21版本，很可能还会看到API 21级里添加的新特性。）

```
<style name="Base.Theme.AppCompat" parent="Base.V7.Theme.AppCompat">
</style>
```

Base.Theme.AppCompat这个主题没任何自己的定义，也就是说没覆盖任何属性。继续定位到它的父主题：Base.V7.Theme. AppCompat。

```
<style name="Base.V7.Theme.AppCompat" parent="Platform.AppCompat">
    <item name="windowNoTitle">false</item>
    <item name="windowActionBar">true</item>
    <item name="windowActionBarOverlay">false</item>
    ...
</style>
```

距离目标越来越近了。Base.V7.Theme.AppCompat有许多属性，但还是没找到改变背景色的属性。继续定位到Platform.AppCompat。这个主题也有多个版本，选择values/values.xml版本。

```
<style name="Platform.AppCompat" parent="android:Theme">
    <item name="android:windowNoTitle">true</item>

    <!-- Window colors -->
    <item  name="android:colorForeground">@color/foreground_material_dark</item>
    <item name="android:colorForegroundinverse">@color/
                              foreground_material_light</item>
    ...
</style>
```

终于，在这里看到了Platform.AppCompat的android:Theme父主题。

注意，这里引用的不是Theme，而是android:Theme。前面的android命名空间不能丢。

AppCompat库可以看作BeatBox应用的一部分。编译项目时，工具会引入AppCompat库和它的一堆Java和XML文件。这些文件已包含在应用里，如同你自己编写的文件。如果想引用AppCompat库里的资源，像Theme.AppCompat这样，直接引用就可以了。

有些主题包含在Android操作系统里，如Theme，引用时必须加上指向归属地的命名空间。在引用Theme主题时，AppCompat库使用了android:Theme这样的形式，这是因为Theme来自于Android操作系统。

总算找到了。在这里，终于可以看到所有可以覆盖的主题属性。当然，还可以继续定位到Theme主题，不过没这个必要。我们想要的属性已经找到了。

查看代码，可以看到windowBackground这个属性。顾名思义，这就是用于主题背景色的属性。

```
<style name="Platform.AppCompat" parent="android:Theme">
    <item name="android:windowNoTitle">true</item>

    <!-- Window colors -->
    <item  name="android:colorForeground">@color/foreground_material_dark</item>
    <item name="android:colorForegroundinverse">@color/
                          foreground_material_light</item>
```

```
        <item  name="android:colorBackground">@color/background_material_dark</item>
        <item name="android:colorBackgroundCacheHint">@color/
                                  abc_background_cache_hint_selector_material_dark</item>
        <item  name="android:disabledAlpha">@dimen/abc_disabled_alpha_material_dark</item>
        <item name="android:backgroundDimAmount">0.6</item>
        <item  name="android:windowBackground">@color/background_material_dark</item>
        ...
    </style>
```

这也是要在BeatBox应用中覆盖的属性。回到styles.xml文件中，覆盖windowBackground这个属性，如代码清单22-10所示。

代码清单22-10 设置窗口背景（res/values/styles.xml）

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="colorPrimary">@color/red</item>
    <item name="colorPrimaryDark">@color/dark_red</item>
    <item name="colorAccent">@color/gray</item>

    <item name="android:windowBackground">@color/soothing_blue</item>
</style>
```

注意，windowBackground这个属性来自Android操作系统，所以别忘了使用android命名空间。

运行BeatBox应用。滚动到recycler视图底部查看背景，没有按钮覆盖的地方是浅蓝色，如图22-7所示。



图22-7 设置了主题背景的BeatBox

想修改应用主题，开发者差不多都要经历刚才查找windowBackground属性的过程。没办法，这些属性没有什么文档可参考，只能去看源代码了。

总结一下，刚才我们定位查看了以下主题：

- Theme.AppCompat
- Base.Theme.AppCompat
- Base.V7.Theme.AppCompat
- Platform.AppCompat

刚才我们自下而上逐层定位，直到找到AppCompat根主题。将来，越来越熟练之后，你很可能会跳过中间步骤而直达目标。不过，建议还是按部就班，以此看清楚究竟哪个是根主题。

最后再提个醒，主题继承关系和层次可能有变（发布新系统），但上面介绍的方法不会变。想要知道该覆盖哪个属性，就沿着继承树找吧！

22.6　修改按钮属性

前面，通过在res/layout/list_item_sound.xml文件中手工设置样式属性，我们定制过BeatBox应用的按钮。如果一个复杂应用有很多fragment，都有很多按钮，再去逐个fragment、逐个按钮地去设置style属性就很不应该了。在这种情况下，还是要靠主题。你可以在主题中定义一个用于所有按钮的样式。

在主题里添加按钮样式前，先打开res/layout/list_item_sound.xml文件，删掉原有样式属性，如代码清单22-11所示。

代码清单22-11　删掉！有更好的办法了（res/layout/list_item_sound.xml）

```
<Button
    style="@style/BeatBoxButton.Strong"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:onClick="@{() -> viewModel.onButtonClicked()}"
    android:text="@{viewModel.title}"
    tools:text="Sound name"/>
```

运行BeatBox应用。可以看到，按钮回到原来的模样了。

再次逐级定位查找主题。这次，我们找到Base.V7.Theme.AppCompat里的buttonStyle属性。

```
<style name="Base.V7.Theme.AppCompat" parent="Platform.AppCompat">
    ...
    <!-- Button styles -->
    <item name="buttonStyle">@style/Widget.AppCompat.Button</item>
    <item name="buttonStyleSmall">@style/Widget.AppCompat.Button.Small</item>
    ...
</style>
```

这个属性指定应用中普通按钮的样式。

这个buttonStyle属性没有设置值，而是指向了一个样式资源。前面覆盖windowBackground属性时，直接传入了颜色值。这里，buttonStyle应该指向另一个样式。定位并查看Widget.AppCompat.Button样式。

```
<style name="Widget.AppCompat.Button" parent="Base.Widget.AppCompat.Button"/>
```

Widget. AppCompat.Button样式没有定义任何属性，继续定位找其指向的父样式。你会发现有两个版本可选，选values/values.xml版本。

```
<style name="Base.Widget.AppCompat.Button" parent="android:Widget">
    <item  name="android:background">@drawable/abc_btn_default_mtrl_shape</item>
    <item  name="android:textAppearance">?android:attr/textAppearanceButton</item>
    <item name="android:minHeight">48dip</item>
    <item name="android:minWidth">88dip</item>
    <item name="android:focusable">true</item>
    <item name="android:clickable">true</item>
    <item name="android:gravity">center_verticallcenter_horizontal</item>
</style>
```


BeatBox应用的所有按钮都使用了这些属性。

在BeatBox应用里复用Android自身主题。修改BeatBoxButton样式的父样式为Widget.AppCompat.Button。另外，删除BeatBoxButton.Strong样式，如代码清单22-12所示。

代码清单22-12　创建按钮样式（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
        <item name="colorPrimary">@color/red</item>
        <item name="colorPrimaryDark">@color/dark_red</item>
        <item name="colorAccent">@color/gray</item>

        <item name="android:windowBackground">@color/soothing_blue</item>
    </style>

    <style name="BeatBoxButton" parent=" Widget.AppCompat.Button">
        <item name="android:background">@color/dark_blue</item>
    </style>

    <style name="BeatBoxButton.Strong">
        <item name="android:textStyle">bold</item>
    </style>

</resources>
```

继承Widget.AppCompat.Button样式，就是首先让所有按钮都继承常规按钮的属性。然后根据需要，有选择性地修改一些属性。

如果不指定BeatBoxButton样式的父样式，所有按钮会变得不再像个按钮，连按钮中间显示的文字都会丢失。

BeatBoxButton样式已重新定义完毕，可以使用了。经过前面主题深挖，我们知道要覆盖buttonStyle属性。下面覆盖buttonStyle属性，让它指向BeatBoxButton样式，如代码清单22-13所示。

代码清单22-13　使用BeatBoxButton样式（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
        <item name="colorPrimary">@color/red</item>
        <item name="colorPrimaryDark">@color/dark_red</item>
        <item name="colorAccent">@color/gray</item>

        <item name="android:windowBackground">@color/soothing_blue</item>
        <item name="buttonStyle">@style/BeatBoxButton</item>
    </style>

    <style name="BeatBoxButton" parent="Widget.AppCompat.Button">
        <item name="android:background">@color/dark_blue</item>
    </style>

</resources>
```

注意，定义buttonStyle时，我们没有使用android:前缀。这是因为，要覆盖的buttonStyle属性是在AppCompat库里实现的。

现在，buttonStyle属性已被覆盖，你使用了自定义的BeatBoxButton。

运行BeatBox应用，所有的按钮都变成了深蓝色了，如图22-8所示。没有直接修改任何布局，就改变了普通按钮的样子。Android主题属性太强大了！



图22-8　带最终版主题的BeatBox

按钮没有轮廓，很不明显。下一章会做美化，让它们更美观。

22.7 深入学习：样式继承拾遗

本章前面对样式继承知识点的介绍还不够全面。在进行主题探秘时，你可能已经注意到了，样式继承的表示法时有切换。AppCompat主题都是使用主题名表示继承，直到碰到`Platform.AppCompat`这个主题。

```
<style name="Platform.AppCompat" parent="android:Theme">
  ...
</style>
```


这里，继承是直接使用`parent`属性来表示的。为什么呢？

要以主题名的形式指定父主题，有继承关系的两个主题都应处于同一个包中。因此，对于Android操作系统内部主题间的继承，就可以直接使用主题名继承表示法。同理，AppCompat库内部也是这样。然而，一旦AppCompat库要跨库继承，就一定要明确使用`parent`属性。

在开发自己的应用时，应遵守同样的规则。如果是继承自己内部的主题，使用主题名指定父主题即可；如果是继承Android操作系统中的样式或主题，记得使用`parent`属性。

22.8 深入学习：引用主题属性

在主题中定义好属性后，可以在XML或代码中直接使用它们。

为了在XML中引用主题属性，我们使用第7章中`divider`属性用到的符号。在XML中引用具体值（如颜色值）时，我们使用@符号。`@color/gray`指向某个特定资源。

在主题中引用资源时，使用?符号。

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:background="?attr/colorAccent"
    tools:text="Sound name"/>
```

上述XML中?符号的意思是使用`colorAccent`属性指向的资源。这里，是指定义在colors.xml文件中的灰色。

也可以在代码中使用主题属性，但是比较啰嗦。

```
Resources.Theme theme = getActivity().getTheme();
int[] attrsToFetch = { R.attr.colorAccent };
TypedArray a = theme.obtainStyledAttributes(R.style.AppTheme, attrsToFetch);
int accentColor = a.getInt(0, 0);
a.recycle();
```

先取得`Theme`对象，然后要求它找到定义在`AppTheme`（即`R.style.AppTheme`）中的`R.attr.colorAccent`属性。结果得到一个持有数据的`TypedArray`对象。接着，向`TypedArray`对象索要`int`值以取出颜色。颜色值取出之后就可以使用了，比如，用来更改按钮背景色。

BeatBox应用中的工具栏和按钮就是采取上述方式使用主题属性美化自己的。

第 23 章 XML drawable

BeatBox应用的主题非常漂亮，下面该是优化按钮表现的时候了。

当前，按钮就是个蓝方框，点击它也看不到任何反应。本章，我们将学习使用XML drawable，继续美化BeatBox应用，让它拥有如图23-1所示的用户界面。



图23-1　完全改观的用户界面

在Android世界里，凡是要在屏幕上绘制的东西都可以叫作drawable，比如抽象图形、`Drawable`类的子类代码、位图图像等。本章，你还会看到更多的drawable：state list drawable、shape drawable和layer list drawable。这三个drawable都定义在XML文件中，可以归为一类，统称为XML drawable。

23.1　统一按钮样式

定义XML drawable之前，先修改list_item_sound.xml文件隔开按钮，如代码清单23-1所示。

代码清单23-1　隔开按钮（res/layout/list_item_sound.xml）

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
```

```
        <data>
            <variable
                name="viewModel"
                type="com.bignerdranch.android.beatbox.SoundViewModel"/>
        </data>
        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="8dp">
            <Button
                android:layout_width="match_parent"
                android:layout_height="120dp"
                android:layout_width="100dp"
                android:layout_height="100dp"
                android:layout_gravity="center"
                android:onClick="@{() -> viewModel.onButtonClicked()}"
                android:text="@{viewModel.title}"
                tools:text="Sound name"/>
        </FrameLayout>
    </layout>
```

现在，按钮的宽和高都是100dp。这样，稍后变为圆形时，这些按钮就不会歪斜了。

不论屏幕大小，recycler视图总是显示三列按钮。如果还有多余的空间，它会拉伸列格以适配屏幕。不过，BeatBox应用的按钮不应拉伸，所以把它们封装在frame布局里。这样，frame布局会被拉伸，而按钮不会。

运行BeatBox应用。按钮的尺寸都完全统一了，并且彼此之间还留出了空间，如图23-2所示。



图23-2 隔开的按钮

23.2 shape drawable

使用`ShapeDrawable`，可以把按钮变成圆。XML drawable和屏幕像素密度无关，所以无需考虑创建特定像素密度目录，直接把它放入默认的drawable文件夹就可以了。

打开项目工具窗口，在res/drawable目录下创建一个名为button_beat_box_normal.xml的文件，如代码清单23-2所示。（稍后还会创建一个“非正常”的文件，所以文件名里有normal字样。）

代码清单23-2　创建圆形drawable（res/drawable/button_beat_box_normal.xml）

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <solid
        android:color="@color/dark_blue"/>

</shape>
```

该XML文件定义了一个背景为深蓝色的圆形。也可使用shape drawable定制其他各种图形，如长方形、线条以及梯形等。欲详细了解shape drawable定制信息，可查看开发者文档：developer.android.com/guide/topics/resources/drawable-resource.html）。

在styles.xml中，使用新建的`button_beat_box_normal`作为按钮背景，如代码清单23-3所示。

代码清单23-3　修改按钮背景（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
      ...
    </style>

    <style name="BeatBoxButton" parent=" Widget.AppCompat.Button">
      <item name="android:background">@color/dark_blue</item>
      <item name="android:background">@drawable/button_beat_box_normal</item>
    </style>

</resources>
```

运行BeatBox应用。可以看到，圆形的按钮出现了，如图23-3所示。



图23-3　圆形按钮

点击按钮之后会听到播放的声音，可按钮的样子却没有任何变化。按钮按下去时，如果能切换显示状态，用户体验应该会更好。

23.3　state list drawable

为解决这个问题，首先定义一个用于按钮按下状态的shape drawable。

在res/drawable目录下再创建一个名为button_beat_box_pressed.xml的文件，如代码清单23-4所示。除了背景颜色是红色外，这个shape drawable和前面的正常版本是一样的。

代码清单23-4　定义按钮按下时的shape drawable（res/drawable/button_beat_box_pressed.xml）

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <solid
        android:color="@color/red"/>

</shape>
```

接下来，要在按钮按下时使用这个新建的shape drawable。这需要用到state list drawable。

根据按钮的状态，state list drawable可以切换指向不同的drawable。按钮没有按下的时候指向`button_beat_box_normal`，按下的时候就指向`button_beat_box_pressed`。

在drawable目录中，定义一个state list drawable，如代码清单23-5所示。

代码清单23-5　创建一个state list drawable（res/drawable/button_beat_box.xml）

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_beat_box_pressed"
        android:state_pressed="true"/>
    <item android:drawable="@drawable/button_beat_box_normal" />
</selector>
```

现在，在styles.xml中修改按钮样式，改用`button_beat_box`作为按钮背景，如代码清单23-6所示。

代码清单23-6　使用state list drawable（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
      ...
    </style>

    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
        <item name="android:background">@drawable/button_beat_box_normal</item>
        <item name="android:background">@drawable/button_beat_box</item>
    </style>

</resources>
```

按钮没有按下的时候使用`button_beat_box_normal`作背景，按下时就使用`button_beat_box_pressed`作背景。

运行BeatBox应用。查看按下状态的按钮背景，如图23-4所示。

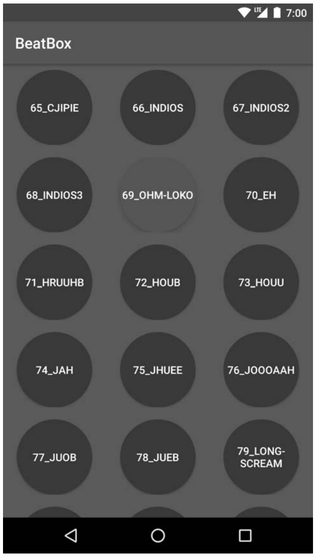


图23-4　按下状态的按钮

除了按下状态，state list drawable还支持禁用、聚焦以及激活等状态。若想详细了解，请访问网页：developer.android.com/guide/topics/resources/drawable-resource.html#StateList。

23.4　layer list drawable

BeatBox应用看起来挺不错了。按钮圆圆的，按下时还有视觉反馈。不过，还要精益求精。

layer list drawable能让两个XML drawable合二为一。借助这个工具，可以为按下状态的按钮添加一个深色的圆环，如代码清单23-7所示。

代码清单23-7　使用layer list drawable（res/drawable/button_beat_box_pressed.xml）

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape xmlns:android="http://schemas.android.com/apk/res/android"
            android:shape="oval">

          <solid
              android:color="@color/red"/>
        </shape>
    </item>
    <item>
        <shape
```

```
            android:shape="oval">

            <stroke
                android:width="4dp"
                android:color="@color/dark_red"/>
        </shape>
    </item>
</layer-list>
```

现在，layer list drawable中指定了两个drawable。第一个是和以前一样的红圈。第二个则会绘制在第一个圈上，它定义了一个4dp粗的深红圈。这会产生一个暗红的圈。

这两个drawable可以组成一个layer list drawable。多个当然也可以，会获得一些更复杂的效果。

运行BeatBox应用，随意点击几个按钮。可以看到，在按下状态，按钮有了漂亮的边圈，如图23-5所示。

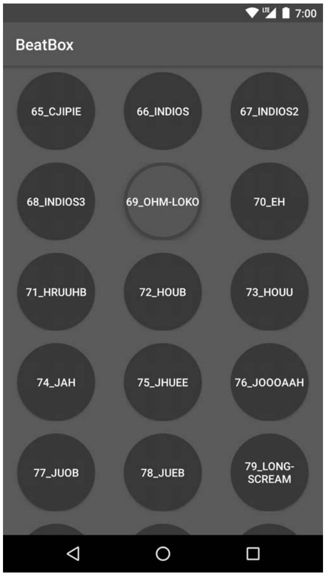


图23-5　最终版BeatBox

现在，BeatBox应用真正完成了。还记得应用最初的样子吗？两相对比，简直云泥之别。实践证明，精美的应用让人用起来舒心，容易获得用户的青睐。

23.5　深入学习：为什么要用 XML drawable

应用总需要切换按钮状态，所以state list drawable是Android开发不可或缺的工具。那shape drawable和layer list drawable呢？应该用吗？

XML drawable用起来方便灵活，不仅用法多样，还易于更新维护。搭配使用shape drawable

和layer list drawable可以做出复杂的背景图，连图像编辑器都省了。更改BeatBox应用的配色更是简单，直接修改XML drawable中的颜色就行了。

另外，XML drawable独立于屏幕像素密度，可在不带屏幕密度资源修饰符的drawable目录中直接定义。如果是普通图像，就需要准备多个版本，以适配不同屏幕像素密度的设备；而XML drawable只要定义一次，就能在任何设备的屏幕上表现出色。

23.6　深入学习：使用 mipmap 图像

资源修饰符和drawable用起来都很方便。应用要用到图像，就针对不同的设备尺寸准备不同尺寸的图片，再分别放入drawable-mdpi和drawable-hdpi这样的文件夹。然后，按名字引用它们。剩下的就交给Android了，它会根据当前设备的屏幕密度调用相应的图片。

但是，有个问题不得不提。发布应用到Google应用商店时，APK文件包含了项目drawable目录里的所有图片。这里面有些图片甚至从来不会用到。这是个负担。

为解决这个问题，有人想到针对设备定制APK，比如mdpi APK一个，hdpi APK一个，等等。（有关APK分包的详细信息，可参阅工具文档网页：tools.android.com/tech-docs/new-build-system/user-guide/apk-splits。）

但问题解决得不够彻底。假如想保留各个屏幕像素密度的启动图标呢?

Android启动器是个常驻主屏幕的应用（详见第24章）。按下设备的主屏幕键，会回到启动器应用界面。

有些新版启动器会显示大尺寸应用图标。想让大图标清晰好看，启动器就需要使用更高分辨率的图标。对于hdpi设备，要显示大图标，启动器就会使用xhdpi图标。找不到的话，就只能使用低分辨率的图标。可想而知，放大拉伸后的图标肯定很糟。

Android的另一解决办法是使用mipmap目录。本书编写时，Android Studio中的新项目已经可以使用mipmap资源作为应用的启动图标了，如图23-6所示。

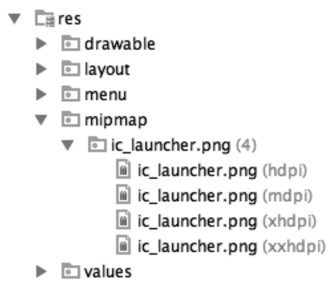


图23-6　mipmap图标

据此，我们有个推荐做法：把应用启动器图标放在mipmap目录中，其他图片都放在drawable目录中。

23.7 深入学习：使用 9-patch 图像

有时候（也可能经常），按钮背景图必须用到普通图片。那么，如果按钮需要以不同尺寸显示，背景图该如何变化呢？

如果按钮的宽度大于背景图的宽度，图片会被拉伸。拉伸的图片会有很好的效果吗？

朝一个方向拉伸背景图很可能会让图片失去原样，所以得想个办法控制图片拉伸方式。

本节，改造BeatBox应用按钮，我们使用9-patch图片做其背景（不明白没关系，稍后就知道了）。注意，之所以改造，并不是说9-patch更适合BeatBox。我们仅仅是想告诉你9-patch是如何工作的，在将来需要的时候，你该如何使用它。

首先，修改list_item_sound.xml文件，允许按钮随屏幕大小动态调整，如代码清单23-8所示。

代码清单23-8　允许拉伸按钮（res/layout/list_item_sound.xml）

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="viewModel"
            type="com.bignerdranch.android.beatbox.SoundViewModel"/>
    </data>
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp">
        <Button
            android:layout_width="100dp match_parent"
            android:layout_height="100dp match_parent"
            android:layout_gravity="center"
            android:onClick="@{() -> viewModel.onButtonClicked()}"
            android:text="@{viewModel.title}"
            tools:text="Sound name"/>
    </FrameLayout>
</layout>
```

调整后，按钮会使用多余空间，按钮的间隔还是8dp。新按钮背景图有个折角和阴影，如图23-7所示。这是按钮的新背景图。



图23-7　新背景图（res/drawable-xxhdpi/ic_button_beat_box_default.png）

在随书文件的xxhdpi drawable目录里（对应本章），找到包括按下状态在内的两个新背景图，复制到BeatBox项目的drawable-xxhdpi目录中。然后修改button_ beat_box.xml文件使用它们，如代码清单23-9所示。

代码清单23-9　使用新背景图（res/drawable/button_beat_box.xml）

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_beat_box_pressed"
    <item android:drawable="@drawable/ic_button_beat_box_pressed"
        android:state_pressed="true"/>
    <item android:drawable="@drawable/button_beat_box_normal"
    <item android:drawable="@drawable/ic_button_beat_box_default"
</selector>
```

运行应用，查看按钮显示效果，如图23-8所示。

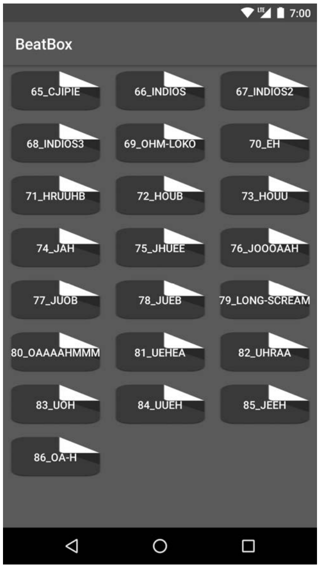


图23-8　难看的背景图

呃，这也太丑了吧！

为什么这么丑？原来，Android向四面拉伸了ic_beat_box_button.png，包括折边和圆角。要是能控制该拉伸的部分拉伸，不该拉伸的不拉伸就好了。

使用9-patch图像能解决这个问题。9-patch图像是一种特别处理过的文件，能让Android知道图像的哪些部分可以拉伸，哪些部分不可以。只要处理得当，就能确保背景图的边角与原始图像保持一致。

为什么要叫作9-patch呢？9-patch图像分成3 × 3的网格，即由9部分或9 patch组成的网格。网格角落部分不会被缩放，边缘部分的4个patch只按一个维度缩放，而中间部分则按两个维度缩放，如图23-9所示。

9-patch图像和普通PNG图像十分相似，只有两处不同：9-patch图像文件名以.9.png结尾，图像边缘具有1像素宽度的边框。这个边框用以指定9-patch图像的中间位置。边框像素绘制为黑线，以表明中间位置，边缘部分则用透明色表示。

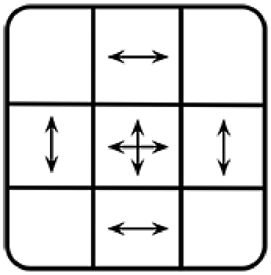


图23-9 9-patch拉伸原理

任意图形编辑器都可用来创建9-patch图像，但Android SDK自带的draw9patch工具用起来更方便。

首先，把两张新背景图转换为9-patch图像。在项目工具窗口中，右键单击ic_button_beat_box_default.png，选择Refactor → Rename...菜单项将其改名为ic_button_beat_box_default.9.png。（如果Android Studio提示有同名资源，直接点Continue按钮继续。）再用相同的步骤得到另一个文件：ic_button_beat_box_pressed.9.png。

然后，双击默认图片在Android Studio内置的9-patch工具中打开，如图23-10所示。（如果Android Studio没能顺利打开9-patch编辑器，请先关闭图片文件，并在项目工具窗口中展开drawable目录，再尝试重新打开它。）

在9-patch工具中，首先，为让图片更醒目，勾选上Show patches选项。然后，把图像顶部和左边框填充为黑色，以标记图像的可伸缩区域（见图23-10）。

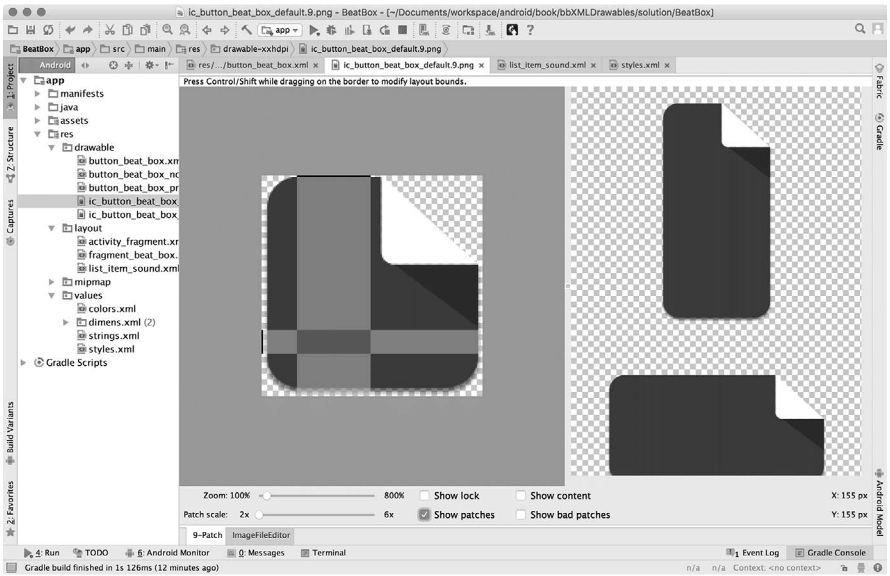


图23-10 创建9-patch图像

图片的顶部黑线指定了水平方向的可拉伸区域。左边的黑线标记在竖直方向哪些像素可以拉伸。

重复上述步骤处理好另一个版本的图像。运行应用，看看9-patch新图是什么效果（见图23-11）。

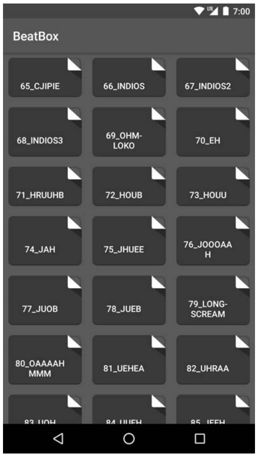


图23-11　9-patch图像使用效果

顶部以及左边框标记了图像的可拉伸区域，那么底部以及右边框又该如何处理呢？它们定义了9-patch图像的可选内容区。内容区是绘制内容（通常是文字）的地方。如果不标记内容区，那么默认与可拉伸区域保持一致。

使用内容区让按钮上的文字居中。现在继续编辑ic_button_beat_box_default.9.png，如图23-12所示，在图片上添加上右边和底部两条线。同时勾选上Show content选项。这个选项会让预览器高亮显示图片的文字显示区。

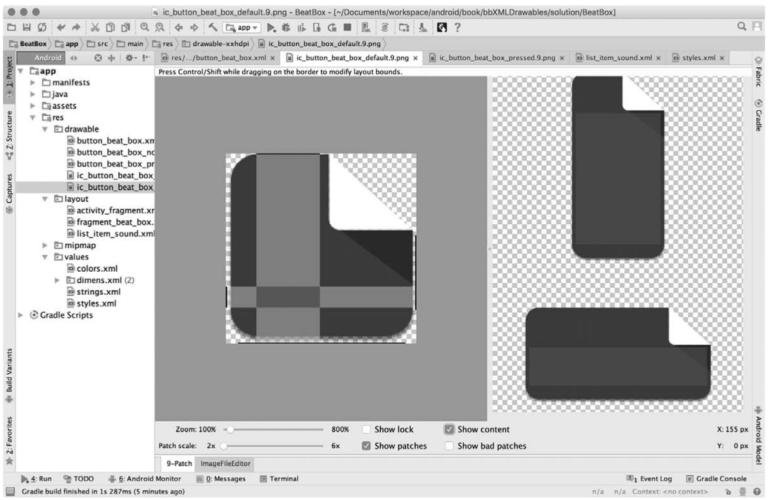


图23-12　定义内容区

重复上述步骤处理好另一个版本的图像。仔细确认两张图像添加的内容区黑线都正确一致。state list drawable使用9-patch图片时（BeatBox应用中），内容区可能会有非预期表现。按钮背景图初始化时，Android会设置内容区内容，而在用户按下按钮时，内容区内容很可能不会有变化。这说明，两张之中有一张图片的内容区未定义。所以，这时就要检查看看，state list drawable使用的所有9-patch图片是否都有相同的内容区。

运行BeatBox应用，可以看到文字都居中显示了，如图23-13所示。



图23-13　BeatBox应用新面貌

试着横屏查看应用。可以看到，图像拉伸得更厉害了，不过按钮背景图的效果依然不错，文字依然能居中显示。

23.8　挑战练习：按钮主题

完成应用9-patch图片更新后，你可能已注意到按钮的背景图有点不对劲：图片折角后面似乎有阴影。你甚至还注意到，只有在Lollipop或更高系统版本上运行应用时，图片折角后面才会出现阴影。

实际上，这个阴影是按钮默认在Lollipop或更高系统版本获得的一种浮层效果。按下按钮时，它会向你的手指靠拢（详见第35章）。

现在，不替换背景图，去掉这个阴影。回顾前面学的主题相关知识，看看这个阴影是怎么产生的。再思考思考：要解决这个问题，有没有其他按钮样式可用（作为`BeatBoxButton`样式的父样式）？

第 24 章

深入学习intent和任务

本章将使用隐式intent创建一个替换Android默认启动器的应用。新建应用名为NerdLauncher，运行画面如图24-1所示。

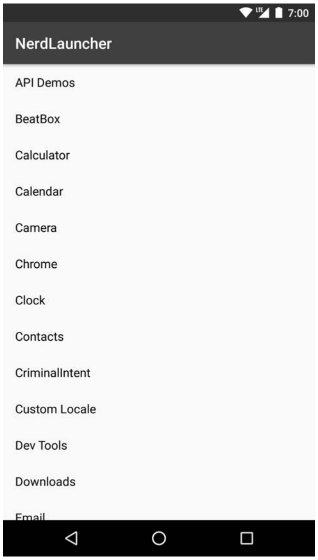


图24-1　NerdLauncher应用最终效果

NerdLauncher应用能列出设备上的其他应用。点选任意列表项会启动相应应用。

完成该应用能帮你深入理解intent、intent过滤器，搞清楚Android应用间是如何交互的。

24.1　创建 NerdLauncher 项目

创建一个Android新项目，取名为NerdLauncher。选择Phone and Tablet作为目标设备，最低SDK版本设为API 19: Android 4.4 (KitKat)。新建名为NerdLauncherActivity的空activity。

NerdLauncherActivity需要托管fragment，所以它也应继承`SingleFragmentActivity`类。找到CriminalIntent项目中的SingleFragmentActivity.java和activity_fragment.xml文件，把它们复制到

NerdLauncher应用项目中备用。

打开NerdLauncherActivity.java文件，修改`NerdLauncherActivity`的超类为`SingleFragmentActivity`类。然后删除默认的模板代码，并覆盖`createFragment()`方法返回一个`NerdLauncherFragment`，如代码清单24-1所示。（`createFragment()`方法会报错。暂时忽略，稍后会创建`NerdLauncherFragment`类解决。）

代码清单24-1　另一个`SingleFragmentActivity`（NerdLauncherActivity.java）

```
public class NerdLauncherActivity extends SingleFragmentActivityAppCompatActivity {

    @Override
    protected Fragment createFragment() {
        return NerdLauncherFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Auto-generated template code... */
    }
}
```


按照第8章所述方法，添加`RecyclerView`依赖项，因为`NerdLauncherFragment`要用它显示应用列表。

为创建fragment布局，重命名layout/activity_nerd_launcher.xml为layout/fragment_nerd_launcher.xml，然后用图24-2中的`RecyclerView`替换原布局内容。



图24-2　创建`NerdLauncherFragment`布局（layout/fragment_nerd_launcher.xml）

最后，以`android.support.v4.app.Fragment`为父类，创建一个名为`NerdLauncherFragment`的新类。在新建类中，新增`newInstance()`方法，覆盖`onCreateView(...)`方法。在覆盖方法中，将`RecyclerView`对象存放在`mRecyclerView`成员变量中，如代码清单24-2所示。（稍后会处理`RecyclerView`的数据绑定。）

代码清单24-2　`NerdLauncherFragment`初始类（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {

    private RecyclerView mRecyclerView;

    public static NerdLauncherFragment newInstance() {
```

```
        return new NerdLauncherFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_nerd_launcher, container, false);
        mRecyclerView = (RecyclerView) v.findViewById(R.id.app_recycler_view);
        mRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        return v;
    }
}
```

运行应用。一切正常的话，可看到如图24-3所示的用户界面。RecyclerView尚未绑定数据，现在还无法看到应用列表。

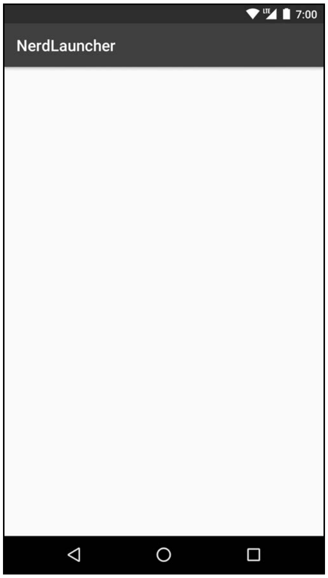


图24-3　NerdLauncher应用初始界面

24.2　解析隐式 intent

NerdLaucher应用会列出设备上的可启动应用。（可启动应用是指点击主屏幕或启动器界面上的图标就能打开的应用。）要实现该功能，它会使用PackageManager获取所有可启动主activity。可启动主activity都带有包含MAIN操作和LAUNCHER类别的intent过滤器。在之前项目的AndroidManifest.xml文件中，你已见过这种intent过滤器：

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
```

```
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
```

在NerdLauncherFragment.java中，新增`setupAdapter()`方法，然后在`onCreateView(...)`方法中调用它。（该方法最终还会创建`RecyclerView.Adapter`实例并设置给`RecyclerView`对象。）另外，再创建一个隐式intent并从`PackageManager`那里获取匹配它的所有activity。最后，记录下`PackageManager`返回的activity总数，如代码清单24-3所示。

代码清单24-3　向`PackageManager`查询activity总数（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {
    private static final String TAG = "NerdLauncherFragment";

    private RecyclerView mRecyclerView;

    public static NerdLauncherFragment newInstance() {
        return new NerdLauncherFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...
        setupAdapter();
        return v;
    }

    private void setupAdapter() {
        Intent startupIntent = new Intent(Intent.ACTION_MAIN);
        startupIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        PackageManager pm = getActivity().getPackageManager();
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);

        Log.i(TAG, "Found " + activities.size() + " activities.");
    }
}
```

运行NerdLauncher应用，在LogCat窗口，看看`PackageManager`返回多少个activity。

在CriminalIntent应用中，为使用隐式intent发送crime报告，我们先创建隐式intent，再将其封装在选择器intent中，最后调用`startActivity(Intent)`方法发送给操作系统：

```
Intent i = new Intent(Intent.ACTION_SEND);
... // Create and put intent extras
i = Intent.createChooser(i, getString(R.string.send_report));
startActivity(i);
```

这里没有使用上述处理方式，是不是很费解？原因很简单：MAIN/LAUNCHER intent过滤器可能无法与通过`startActivity(...)`方法发送的MAIN/LAUNCHER隐式intent相匹配。

事实上，`startActivity(Intent)`方法意味着“启动匹配隐式intent的默认activity”，而不是想当然的“启动匹配隐式intent的activity”。调用`startActivity(Intent)`方法（或`startActivity-`

ForResult(...)方法）发送隐式intent时，操作系统会悄悄为目标intent添加Intent.CATEGORY_DEFAULT类别。

因此，如果希望intent过滤器匹配startActivity(...)方法发送的隐式intent，就必须在对应的intent过滤器中包含DEFAULT类别。

定义了MAIN/LAUNCHER intent过滤器的activity是应用的主要入口点。它只负责做好作为应用主要入口点要处理的工作。它通常不关心自己是否为默认的主要入口点，所以可以不包含CATEGORY_DEFAULT类别。

前面说过，MAIN/LAUNCHER intent过滤器并不一定包含CATEGORY_DEFAULT类别，因此不能保证可以与startActivity(...)方法发送的隐式intent匹配。所以，我们转而使用intent直接向PackageManager查询带有MAIN/LAUNCHER intent过滤器的activity。

接下来，需要在NerdLauncherFragment的RecyclerView视图中显示查询到的activity标签。activity标签是用户可以识别的展示名称。既然查询到的activity都是启动activity，标签名通常也就是应用名。

在PackageManager返回的ResolveInfo对象中，可以获取activity标签和其他一些元数据。

首先，使用ResolveInfo.loadLabel(PackageManager)方法，对ResolveInfo对象中的activity标签按首字母排序，如代码清单24-4所示。

代码清单24-4　对activity标签排序（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {
    ...
    private void setupAdapter() {
        ...
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);
        Collections.sort(activities, new Comparator<ResolveInfo>() {
            public int compare(ResolveInfo a, ResolveInfo b) {
                PackageManager pm = getActivity().getPackageManager();
                return String.CASE_INSENSITIVE_ORDER.compare(
                        a.loadLabel(pm).toString(),
                        b.loadLabel(pm).toString());
            }
        });
        Log.i(TAG, "Found " + activities.size() + " activities.");
    }
}
```

然后，定义一个ViewHolder用来显示activity标签名。另外，ResolveInfo信息经常要用，这里使用成员变量存储它，如代码清单24-5所示。

代码清单24-5　实现ViewHolder（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {
    ...
    private void setupAdapter() {
        ...
    }
```

```
    private class ActivityHolder extends RecyclerView.ViewHolder {
        private ResolveInfo mResolveInfo;
        private TextView mNameTextView;

        public ActivityHolder(View itemView) {
            super(itemView);
            mNameTextView = (TextView) itemView;
        }

        public void bindActivity(ResolveInfo resolveInfo) {
            mResolveInfo = resolveInfo;
            PackageManager pm = getActivity().getPackageManager();
            String appName = mResolveInfo.loadLabel(pm).toString();
            mNameTextView.setText(appName);
        }
    }
}
```


接下来实现RecyclerView.Adapter，如代码清单24-6所示。

代码清单24-6 实现RecyclerView.Adapter（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {
    ...
    private class ActivityHolder extends RecyclerView.ViewHolder {
        ...
    }

    private class ActivityAdapter extends RecyclerView.Adapter<ActivityHolder> {
        private final List<ResolveInfo> mActivities;

        public ActivityAdapter(List<ResolveInfo> activities) {
            mActivities = activities;
        }

        @Override
        public ActivityHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            LayoutInflater layoutInflater = LayoutInflater.from(getActivity());
            View view = layoutInflater
                    .inflate(android.R.layout.simple_list_item_1, parent, false);
            return new ActivityHolder(view);
        }

        @Override
        public void onBindViewHolder(ActivityHolder holder, int position) {
            ResolveInfo resolveInfo = mActivities.get(position);
            holder.bindActivity(resolveInfo);
        }

        @Override
        public int getItemCount() {
            return mActivities.size();
        }
```

```
        }
    }
```

最后，更新`setupAdapter()`方法，创建一个`ActivityAdapter`实例并配置给`RecyclerView`，如代码清单24-7所示。

代码清单24-7　为`RecyclerView`设置adapter（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {
    ...
    private void setupAdapter() {
        ...
        Log.i(TAG, "Found " + activities.size() + " activities.");
        mRecyclerView.setAdapter(new ActivityAdapter(activities));
    }
    ...
}
```

运行NerdLauncher应用。现在可以看到显示了activity标签的`RecyclerView`视图，如图24-4所示。

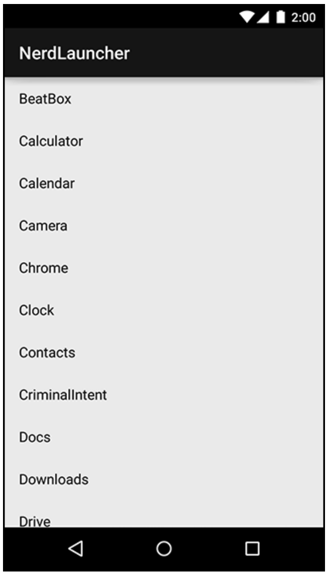


图24-4　设备上的全部activity

24.3　在运行时创建显式 intent

上一节，我们使用隐式intent获取目标activity并以列表的形式展示。接下来要实现点击任一列表项时，启动对应的activity。这次，我们需要使用显式intent来启动activity。

要创建启动activity的显式intent，需要从`ResolveInfo`对象中获取activity的包名与类名。这

些信息可以从ResolveInfo对象的ActivityInfo中获取。（从ResolveInfo类中还可以获取其他信息，具体请查阅该类的参考文档：developer.android.com/reference/android/content/pm/ResolveInfo.html。）

更新ActivityHolder类实施一个点击监听器。然后，使用ActivityInfo对象中的数据信息，创建一个显式intent并启动目标activity，如代码清单24-8所示。

代码清单24-8 启动目标activity（NerdLauncherFragment.java）

```
private class ActivityHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    private ResolveInfo mResolveInfo;
    private TextView mNameTextView;

    public ActivityHolder(View itemView) {
        super(itemView);
        mNameTextView = (TextView) itemView;
        mNameTextView.setOnClickListener(this);
    }

    public void bindActivity(ResolveInfo resolveInfo) {
        ...
    }

    @Override
    public void onClick(View v) {
        ActivityInfo activityInfo = mResolveInfo.activityInfo;

        Intent i = new Intent(Intent.ACTION_MAIN)
                .setClassName(activityInfo.applicationInfo.packageName,
                        activityInfo.name);

        startActivity(i);
    }
}
```


注意，作为显式intent的一部分，我们还发送了ACTION_MAIN操作。发送的intent是否包含操作，对于大多数应用来说没有什么差别。不过，有些应用的启动行为可能会有所不同。取决于不同的启动要求，同样的activity可能会显示不同的用户界面。开发人员最好能明确启动意图，以便让activity完成它应该完成的任务。

在代码清单24-8中，使用包名和类名创建显式intent时，我们使用了以下Intent方法：

```
public Intent setClassName(String packageName, String className)
```

这不同于以往创建显式intent的方式。之前，我们使用的是接受Context和Class对象的Intent构造方法：

```
public Intent(Context packageContext, Class<?> cls)
```

该构造方法使用传入的参数来获取Intent需要的ComponentName。ComponentName由包名和类名共同组成。传入Activity和Class创建Intent时，构造方法会通过Activity类自行确定

全路径包名。

也可以自己通过包名和类名创建ComponentName，然后使用下面的Intent方法创建显式intent：

```
public Intent setComponent(ComponentName component)
```

不过，setClassName(...)方法能够自动创建组件名，用它可以少写不少代码呢。

运行NerdLauncher应用并尝试启动一些应用。

24.4　任务与回退栈

应用运行时，Android使用任务来跟踪用户的状态。通过Android默认启动器应用打开的应用都有自己的任务。然而，这并不适用于NerdLaucher应用。在NerdLaucher应用中启动的应用怎样获得这样的行为呢？别着急，我们先来搞清楚究竟什么是任务，它是如何工作的。

任务是一个activity栈。栈底部的activity通常称为基activity。栈顶的activity用户能看得到。如果按后退键，栈顶activity会弹出栈外。如果用户看到的是基activity，按后退键，系统就会回到主屏幕。

默认情况下，新activity都在当前任务中启动。在CriminalIntent应用中，无论何时启动新activity，它都会被添加到当前任务中。即使要启动的activity不属于CriminalIntent应用（启动其他activity发送crime报告），它同样也在当前任务中启动。如图24-5所示。

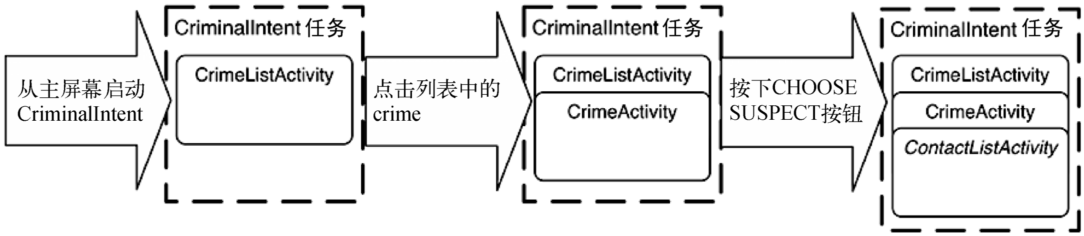


图24-5　CriminalIntent任务

在当前任务中启动activity的好处是，用户可以在任务内而不是在应用层级间导航返回，如图24-6所示。

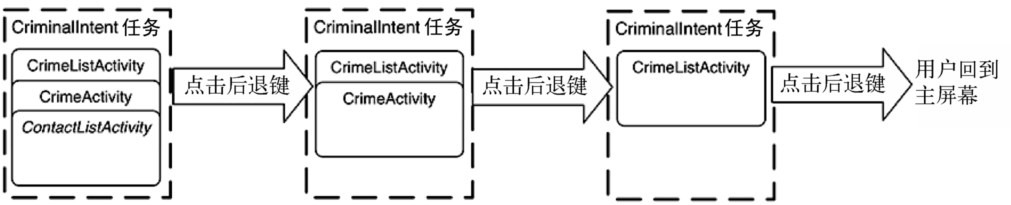


图24-6　按后退键

24.4.1　在任务间切换

在不影响各个任务状态的情况下，overview screen可以让我们在任务间切换（第3章已初识）。例如，一开始你在录联系人信息，然后转到Twitter应用看信息，这时就启动了两个任务。如果再回到联系人应用，你在两个任务中所处的操作状态都会被保存下来。

耳闻不如亲见，你可以在设备或模拟器上试试overview screen。首先，从主屏幕或应用启动器中启动CriminalIntent应用。（如果已卸载，请打开Android Studio中的CriminalIntent项目并运行。）从crime列表中选择任意列表项，然后，按主屏幕键回到主屏幕。接着，从主屏幕或应用启动器中启动BeatBox应用。

现在，按Recents按钮打开overview screen，如图24-7所示。

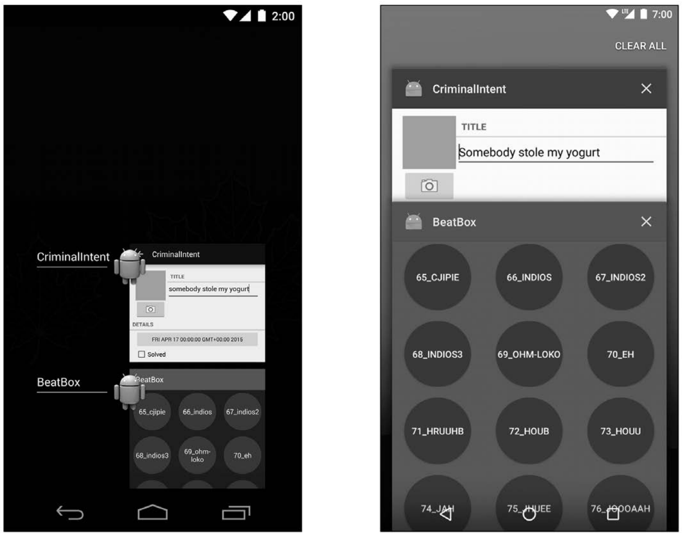


图24-7　overview screen的不同版本

如果是KitKat设备，用户会看到图24-7左边的overview screen；如果是Lollipop，会看到右边的overview screen。不管怎样，图中的每个应用显示项（就是Lollipop系统所说的卡片）就代表着一个应用任务。当前任务显示的是处于回退栈顶部activity的快照。你可以点击任意显示项切换至对应应用的当前activity。

要清除应用任务，用户只需滑动移除卡片即可。清除任务会从应用回退栈中清除所有activity。

试着清除CriminalIntent应用任务再重启。重启后，你看到的是crime列表界面，而不应再是清除前的crime编辑界面了。

24.4.2　启动新任务

有时你需要在当前任务中启动activity，而有时又需要在新任务中启动activity。

当前，从NerdLauncher启动的任何activity都会添加到NerdLauncher任务中，如图24-8所示。

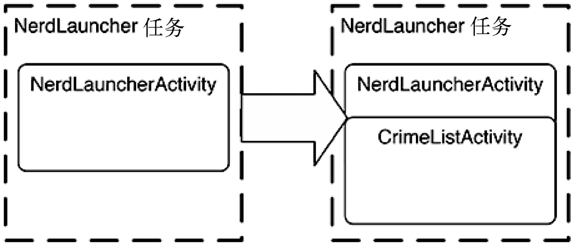


图24-8　NerdLauncher任务中包含CriminalIntent应用activity

要想确认这点，可先清除overview screen显示的所有任务。然后，启动NerdLauncher并点击CriminalIntent列表项启动CriminalIntent应用。再次打开overview screen界面时，应该看不到CriminalIntent任务了。`CrimeListActivity`启动后，它随即就添加到NerdLauncher任务中了，如图24-9所示。只要点击NerdLauncher任务，你就会回到启动overview screen界面之前所在的CriminalIntent用户界面。

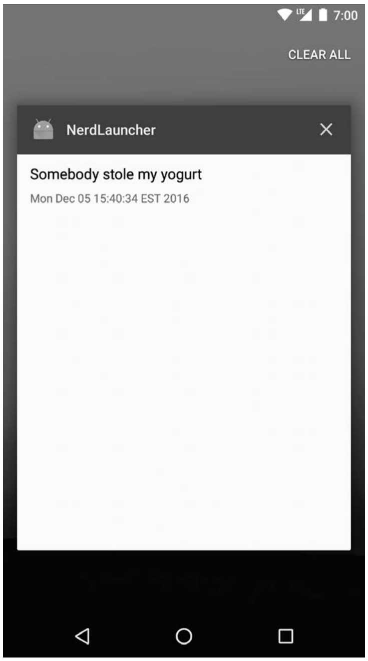


图24-9　CriminalIntent应用在NerdLauncher任务中

我们需要NerdLauncher在新任务中启动activity，如图24-10所示。这样，点击NerdLauncher启动器中的应用项可以让应用拥有自己的任务，用户就可以在运行的应用间自由切换了。

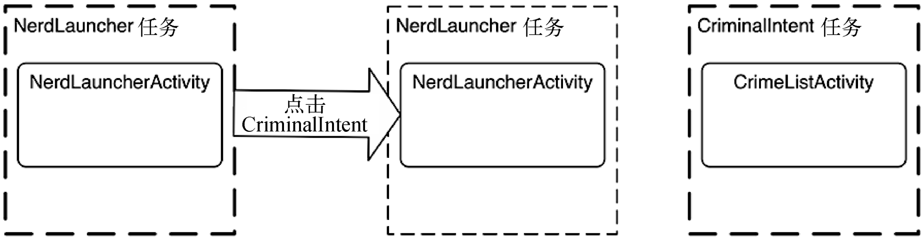


图24-10 让CriminalIntent在自身任务里启动

为了在启动新activity时启动新任务，需要为intent添加一个标志，如代码清单24-9所示。

代码清单24-9 为intent添加新任务标志（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {
    ...
    private class ActivityHolder extends RecyclerView.ViewHolder
            implements View.OnClickListener {
        ...
        @Override
        public void onClick(View v) {
            ...
            Intent i = new Intent(Intent.ACTION_MAIN)
                    .setClassName(activityInfo.applicationInfo.packageName,
                            activityInfo.name)
                    .addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

            startActivity(i);
        }
    }
    ...
}
```


先清除overview screen显示的所有任务，再次运行NerdLauncher应用并启动CriminalIntent。这次，如果启动overview screen，就会看到CriminalIntent应用处于一个单独的任务中，如图24-11所示。

如果从NerdLauncher应用中再次启动CriminalIntent应用，也不会创建第二个CriminalIntent任务。FLAG_ACTIVITY_NEW_TASK标志控制每个activity仅创建一个任务。CrimeListActivity已经有了一个运行的任务，因此Android会自动切换到原来的任务，而不是创建全新的任务。

眼见为实。在CriminalIntent应用中，打开任意crime的明细界面。然后，使用overview screen切换至NerdLauncher。点击应用列表中的CriminalIntent。可以看到，CriminalIntent应用中打开的crime明细界面又回来了。

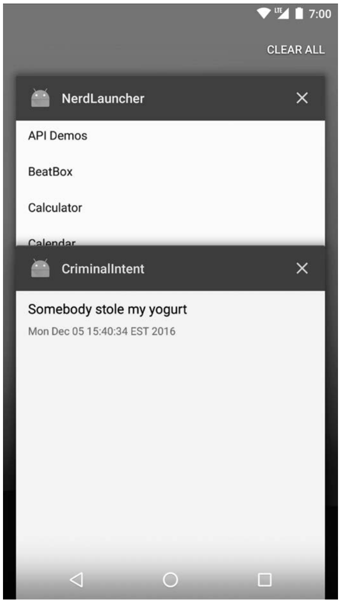


图24-11　CriminalIntent应用处于独立的任务中

24.5　使用 NerdLauncher 应用作为设备主屏幕

没人愿意通过启动一个应用来启动其他应用。因此，以替换Android主界面（home screen）的方式使用NerdLauncher应用会更合适一些。打开NerdLauncher项目的AndroidManifest. xml，向intent主过滤器添加以下节点定义，如代码清单24-10所示。

代码清单24-10　修改NerdLauncher应用的类别（AndroidManifest.xml）

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    <category android:name="android.intent.category.HOME" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

添加HOME和DEFAULT类别定义后，NerdLauncher应用的activity会成为可选的主界面。按主屏幕键可以看到，在弹出的对话框中，NerdLauncher变成了主界面可选项，如图24-12所示。

（如果已设置NerdLauncher应用为主界面，恢复系统默认设置也很容易。首先，从NerdLauncher启动Settings应用。如果是Lollipop系统，可选择Settings → Apps菜单项，然后从应用列表中选择NerdLauncher。如果是Lollipop之前的系统，可以选择Settings→ Applications→Manage Applications菜单项，找到NerdLauncher应用。选择了NerdLauncher后，在应用的信息屏，滚动到Launch by default并点击CLEAR DEFAULTS按钮。好了，下次再按主屏幕键时，又可以自主选了。）

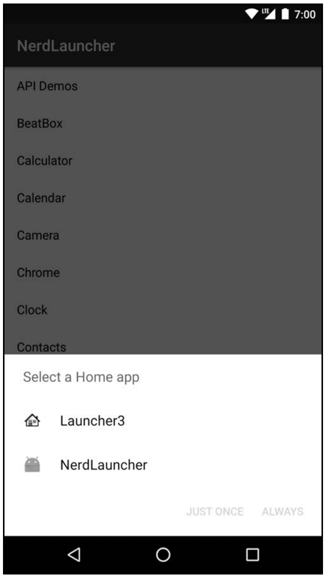


图24-12　选择主屏幕应用

24.6　挑战练习：应用图标

前面，为在启动器应用中显示各个activity的名称，你使用了`ResolveInfo.loadLabel(...)`方法。`loadIcon()`是`ResolveInfo`类的另一个方法，可以用它为每个应用加载显示图标。作为练习，请给NerdLauncher应用中显示的所有应用添加图标。

24.7　深入学习：进程与任务

对象需要内存和虚拟机的支持才能生存。进程是操作系统创建的、供应用对象生存以及应用运行的地方。

进程通常会占用由操作系统管理着的系统资源，如内存、网络端口以及打开的文件等。进程还拥有至少一个（可能多个）执行线程。在Android系统中，每个进程都需要一个虚拟机来运行。

尽管存在未知的异常情况，但总的来说，Android世界里的每个应用组件都仅与一个进程相关联。应用伴随着自己的进程一起完成创建，该进程同时也是应用中所有组件的默认进程。

（虽然组件可以指派给不同的进程，但我们推荐使用默认进程。如果确实需要在不同进程中运行应用组件，通常也可以借助多线程来实现。相比多进程的使用，Android多线程的使用更加简单。）

每一个activity实例都仅存在于一个进程之中，同一个任务关联。这也是进程与任务的唯一相似之处。任务只包含activity，这些activity通常来自于不同的应用进程；而进程则包含了应用的全

部运行代码和对象。

进程与任务很容易让人混淆，主要原因在于它们不仅在概念上有某种重叠，而且通常会被人以应用名提及。例如，从NerdLauncher启动器中启动CriminalIntent应用时，操作系统创建了一个CriminalIntent进程以及一个以`CrimeListActivity`为基栈activity的新任务。在overview screen中，可以看到这个任务就被标名为CriminalIntent。

包含activity的任务和它赖以生存的进程有可能会不同。以CriminalIntent应用和联系人应用为例，看看以下具体场景就会明白了。

打开CriminalIntent应用，选择任意crime项（或添加一条crime记录），然后点击CHOOSE SUSPECT按钮。这会打开联系人应用让你选择目标联系人。随即，联系人列表activity会被加入CriminalIntent应用任务中。如果此时按后退键在不同activity间切换的话，用户可能意识不到他们正在进程间切换。

然而，联系人activity实例实际是在联系人应用进程的内存空间创建的，而且也是在该应用进程里的虚拟机上运行的，如图24-13所示。

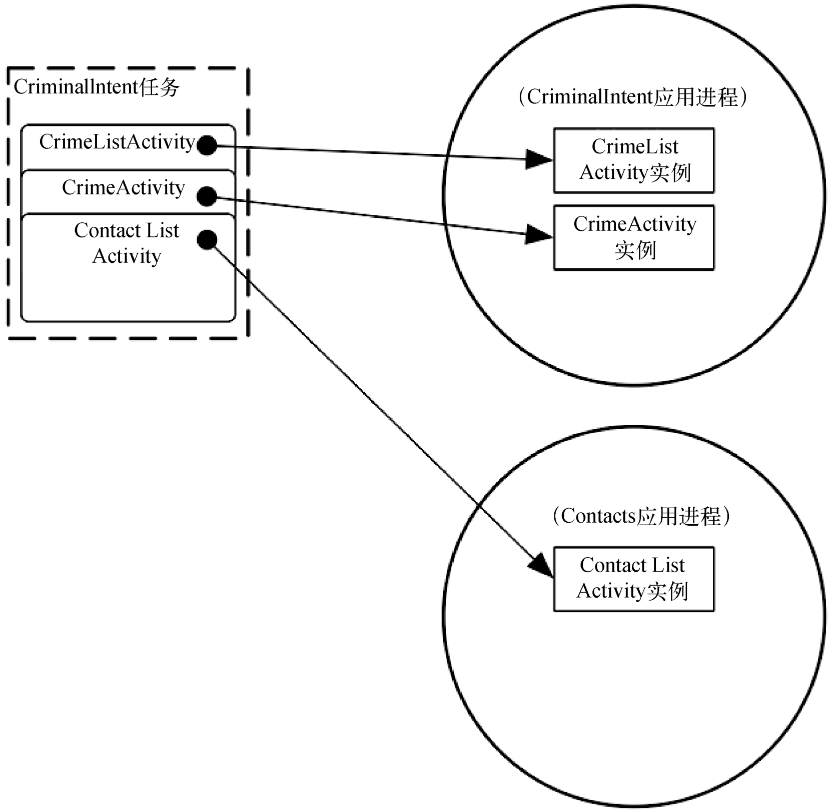


图24-13　任务与进程一对多的关系

为进一步了解进程和任务的概念，让CriminalIntent应用运行着，进入联系人列表界面。（继续之前，请确保在overview screen里看不到联系人应用。）按主屏幕键回到主屏幕，从中启动联系人应用。然后从联系人列表选取任意联系人，或添加新联系人。

在这个操作过程中，系统会在联系人应用进程中创建新的联系人列表activity和联系人明细界面实例。也会创建联系人应用新任务。这个新任务会引用联系人列表和联系人明细界面activity实例，如图24-14所示。

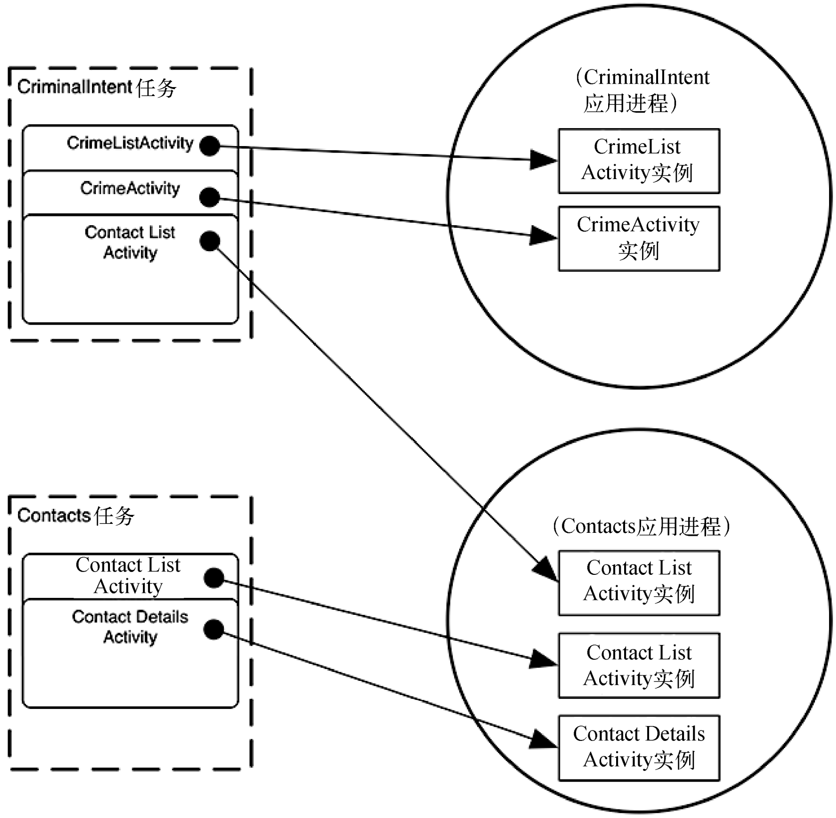


图24-14　进程对多个任务

本章，我们创建了任务并实现了任务间的切换。有没有想过替换Android默认的overview screen呢？很遗憾，做不到，Android没告诉我们怎么做。另外，你应该知道，Google Play商店中那些自称为任务终止器的应用，实际上都是进程终止器。

24.8　深入学习：并发文档

如果是在Lollipop设备上运行CriminalIntent应用，打开overview screen查看任务时，你会发现一些有趣的现象。例如，在发送crime消息时，你所选择发送消息应用的activity不会添加到

CriminalIntent应用任务中，而是添加到它自己的独立任务中，如图24-15所示。

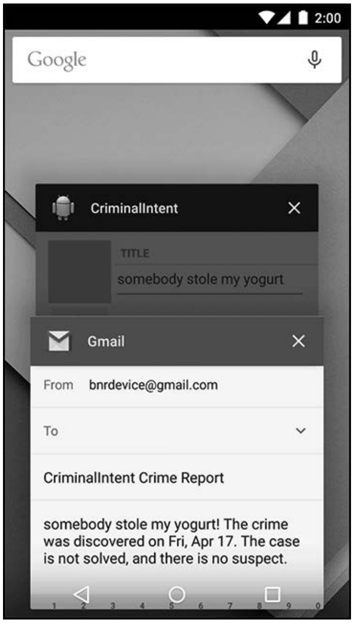


图24-15　Gmail处于独立的任务中

在Lollipop设备上，对以`android.intent.action.SEND`或`action.intent.action.SEND_MULTIPLE`启动的activity，隐式intent选择器会创建独立的新任务。（在旧设备上，Gmail的activity直接添加给了CriminalIntent应用任务。）

这种现象要归因于Lollipop中叫作并发文档（concurrent document）的新概念。有了并发文档，就可以为运行的应用动态创建任意数目的任务。在Lollipop之前，应用任务只能预先定义好，而且还要在manifest文件中指明。

Google Drive就是并发文档概念应用的最好实例。用户可以用它打开并编辑多份文档。这些文档编辑activity都处在独立的任务中，如图24-16所示。在Lollipop之前的设备上查看overview screen的话，你只能看到孤零零的一个任务。前面已说过，Lollipop之前的系统需要在manifest中提前定义应用任务，所以系统无法为单个应用动态创建多个任务。

在Lollipop设备上，如果需要应用启动多个任务，可采用两种方式：给intent打上`Intent.FLAG_ACTIVITY_NEW_DOCUMENT`标签，再调用`startActivity(...)`方法；或者在manifest文件中，为activity设置如下`documentLaunchMode`：

```
<activity
    android:name=".CrimePagerActivity"
    android:label="@string/app_name"
    android:parentActivityName=".CrimeListActivity"
    android:documentLaunchMode="intoExisting" />
```

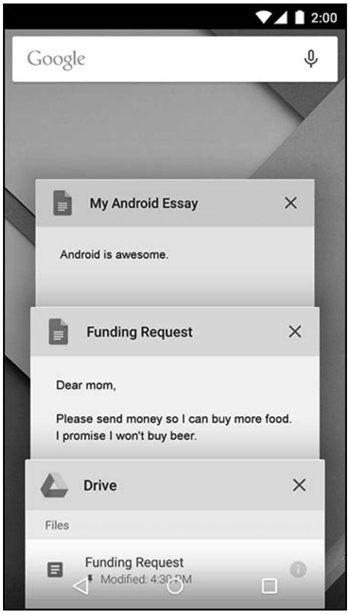


图24-16 Lollipop设备上的多个Google Drive任务

使用上述方法，一份文档只会对应一个任务。（如果发送带有和已存在任务相同数据的intent，系统就不会再创建新任务。）如果无论如何都想创建新任务，那就给intent同时打上`Intent.FLAG_ACTIVITY_NEW_DOCUMENT`和`Intent.FLAG_ACTIVITY_MULTIPLE_TASK`标签，或者把manifest文件中的`documentLaunchMode`属性值改为`always`。

想要深入了解overview screen或者想知道Lollipop系统相较旧系统有哪些变化，请访问网页developer.android.com/guide/components/recents.html。

第 25 章

HTTP与后台任务

信息时代，互联网应用占用了用户的大量时间。餐桌上无人交谈，每个人都只顾低头摆弄手机。一有时间，人们就查看新闻推送、收发短信息，或是玩网络游戏。

为学习Android网络应用的开发，我们来创建一个名为PhotoGallery的应用。PhotoGallery是图片共享网站Flickr的一个客户端应用，它能获取并展示Flickr网站的最新公共图片。应用的最终运行效果如图25-1所示。



图25-1　PhotoGallery应用最终效果图

（PhotoGallery应用有过滤功能，只能展示无版权限制图片。可访问网址www.flickr.com/commons/usage/，进一步了解非限制使用图片。Flickr网站上有很多图片归上传者私有，使用它们需遵守使用许可限制条款。可访问网址www.flickr.com/creativecommons/，了解更多有关第三

方内容的使用权限问题。）

接下来的六章我们都会学习开发PhotoGallery应用。前两章学习网络下载、JSON文件解析、图像显示等基本知识。随后的几章里，会为应用添加一些特色功能，并借此学习搜索、服务、通知、广播接收器以及网页视图等知识。

本章，我们首先学习应用级HTTP网络编程。当前，几乎所有网络服务的开发都是以HTTP网络协议为基础的。本章结束时，你应完成的任务是：获取、解析以及显示Flickr图片的标题，如图25-2所示。（第26章会学习图片获取与显示的相关内容。）

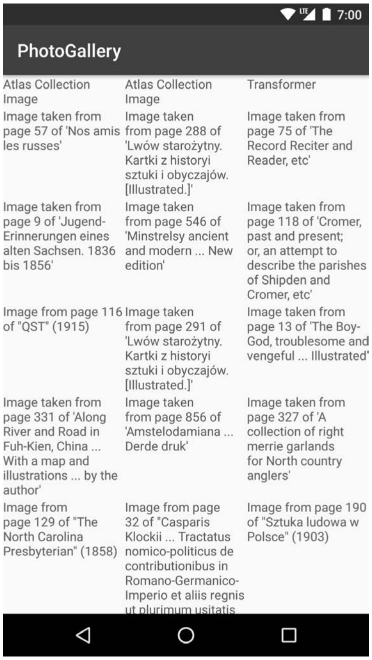


图25-2　本章结束时的应用效果图

25.1　创建 PhotoGallery 应用

按照图25-3所示的配置，创建一个全新的Android应用项目（目标设备选Phone and Tablet，最低SDK版本选API 19）。

单击Next按钮，让应用向导创建一个名为PhotoGalleryActivity的空activity。

PhotoGallery应用继续沿用前面一直使用的设计架构。让`PhotoGalleryActivity`继承`SingleFragmentActivity`，其视图为activity_fragment.xml中定义的容器视图。它会负责托管稍后会创建的`PhotoGalleryFragment`实例。

从之前项目将SingleFragmentActivity.java和activity_fragment.xml复制到当前项目中备用。

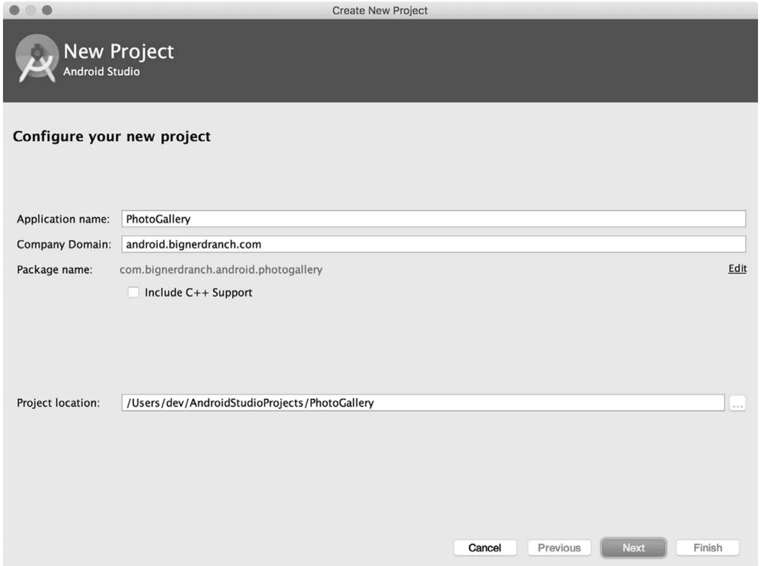


图25-3　创建PhotoGallery应用

在PhotoGalleryActivity.java中，删除工具自动产生的模板代码。然后，让`PhotoGalleryActivity`继承`SingleFragmentActivity`，并实现它的`createFragment()`方法。`createFragment()`方法将返回一个`PhotoGalleryFragment`类实例，如代码清单25-1所示。（有错误提示没关系，稍后创建完`PhotoGalleryFragment`类就好了。）

代码清单25-1　activity的调整（PhotoGalleryActivity.java）

```
public class PhotoGalleryActivity extends Activity SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return PhotoGalleryFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Auto-generated template code... */
    }

}
```

PhotoGallery应用会在`RecyclerView`视图（借助其内置的`GridLayoutManager`）中显示网格内容。

首先是添加RecyclerView依赖项。打开项目结构窗口，选择左边的app模块。再选择Dependencies选项页，单击+按钮。在随后出现的下拉菜单中选择Library dependency。找到并选择recyclerview-v7后点击OK按钮完成。

为创建fragment布局，重命名layout/activity_photo_gallery.xml为layout/fragment_photo_gallery.xml。然后以图25-4所示的RecyclerView组件定义替换原内容。



图25-4　RecyclerView视图（layout/fragment_photo_gallery.xml）

最后，创建PhotoGalleryFragment类，设置其为保留fragment，实例化生成新建布局并引用RecyclerView视图，如代码清单25-2所示。

代码清单25-2　一些代码片断（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {

    private RecyclerView mPhotoRecyclerView;

    public static PhotoGalleryFragment newInstance() {
        return new PhotoGalleryFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

        mPhotoRecyclerView = (RecyclerView) v.findViewById(R.id.photo_recycler_view);
        mPhotoRecyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));

        return v;
    }
}
```

（知道为什么要保留fragment吗？自己先思考一下，答案会在25.7节揭晓。）

继续之前，试着运行PhotoGallery应用。一切正常的话，可以看到一个空白视图。

25.2　网络连接基本

PhotoGallery应用中，我们需要一个网络连接专用类。应用要访问的是Flickr网站，因此新建

一个名为FlickrFetchr的Java类。

FlickrFetchr类一开始只有getUrlBytes(String)和getUrlString(String)两个方法。getUrlBytes(String)方法能从指定URL获取原始数据并返回一个字节流数组。getUrlString(String)方法则将getUrlBytes(String)方法返回的结果转换为String。

在FlickrFetchr.java中，实现getUrlBytes(String)和getUrlString (String)方法，如代码清单25-3所示。

代码清单25-3　基本网络连接代码（FlickrFetchr.java）

```
public class FlickrFetchr {
    public byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();

        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();

            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                throw new IOException(connection.getResponseMessage() +
                        ": with " +
                        urlSpec);
            }

            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while ((bytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, bytesRead);
            }
            out.close();
            return out.toByteArray();
        } finally {
            connection.disconnect();
        }
    }

    public String getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}
```

在getUrlBytes(String)方法中，首先根据传入的字符串参数，如https://www.bignerdranch.com，创建一个URL对象。然后调用openConnection()方法创建一个指向要访问URL的连接对象。URL.openConnection()方法默认返回的是URLConnection对象，但要连接的是http URL，因此需将其强制类型转换为HttpURLConnection对象。这让我们得以调用它的getInputStream()、getResponseCode()等方法。

虽然HttpURLConnection对象提供了一个连接，但只有在调用getInputStream()方法时（如果是POST请求，则调用getOutputStream()方法），它才会真正连接到指定的URL地址，才

会给你反馈代码。

创建了URL并打开网络连接之后，便可循环调用read()方法读取网络数据，直到取完为止。只要还有数据，InputStream类就会不断地输出字节流数据。数据全部取回后，关闭网络连接，并将他们写入ByteArrayOutputStream字节数组中。

虽然最重要的数据获取任务要靠getUrlBytes(String)方法完成，但getUrlString(String)才是本章真正需要的方法。它负责将getUrlBytes(String)方法获取的字节数据转换为String。看到这里，可能有人会问，为什么不在一个方法中完成全部任务？当然可以，但是在下一章处理图像数据下载时，你就能体会两个独立方法的好处了。

获取网络使用权限

要连接网络，还需完成一件事：取得使用网络的权限。正如用户怕被偷拍一样，他们也不想应用偷偷下载图片。

要取得网络使用权限，先要在AndroidManifest.xml文件中添加它，如代码清单25-4所示。

代码清单25-4　在配置文件中添加网络使用权限（AndroidManifest.xml）

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery" >

    <uses-permission android:name="android.permission.INTERNET" />

    <application
      ...
    </application>

</manifest>
```

用户下载应用时（比如PhotoGallery），会看到一个注明需要网络连接权限的对话框，用户可以选择接受或拒绝安装。

如今，大部分应用都需要联网，所以，Android视INTERNET权限为非危险性权限。这样一来，你只要在manifest文件里做个声明，就可以直接使用它了。而有些危险性权限（如获取设备地理位置信息权限），既需要声明又需要运行时动态申请（详见第33章）。

25.3　使用 AsyncTask 在后台线程上运行代码

接下来我们来调用并测试新添加的网络连接代码。注意，不要直接在PhotoGallery-Fragment类中调用FlickrFetchr.getURLString(String)方法。正确的做法是，创建一个后台线程，然后在该线程中运行代码。

使用后台线程最简便的方式是使用AsyncTask工具类。AsyncTask创建后台线程后，我们便可在该线程上调用doInBackground(...)方法运行代码。

在PhotoGalleryFragment.java中，添加一个名为FetchItemsTask的内部类。覆盖AsyncTask.doInBackground(...)方法，从目标网站获取数据并记录日志，如代码清单25-5所示。

代码清单25-5　实现AsyncTask工具类方法，第一部分（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr()
                        .getUrlString("https://www.bignerdranch.com");
                Log.i(TAG, "Fetched contents of URL: " + result);
            } catch (IOException ioe) {
                Log.e(TAG, "Failed to fetch URL: ", ioe);
            }
            return null;
        }
    }
}
```

然后，在PhotoGalleryFragment.onCreate(...)方法中，调用FetchItemsTask新实例的execute()方法，如代码清单25-6所示。

代码清单25-6　实现AsyncTask工具类方法，第二部分（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        new FetchItemsTask().execute();
    }
    ...
}
```

调用execute()方法会启动AsyncTask，进而触发后台线程并调用doInBackground(...)方法。运行PhotoGallery应用，查看LogCat窗口，可以看到一大堆Big Nerd Ranch网站主页HTML代码，如图25-5所示。

LogCat窗口中的日志可不好找。要养成使用关键词搜索的好习惯。图25-5就是在LogCat搜索框中输入PhotoGalleryFragment后的查找结果。

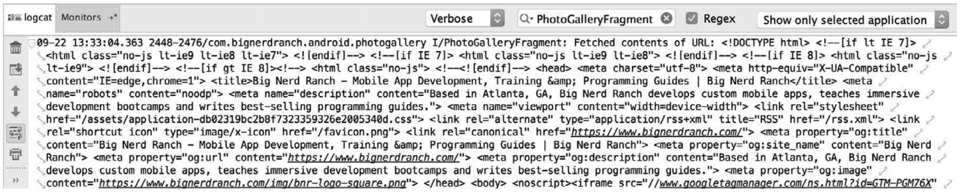


图25-5　LogCat中的HTML代码

既然已创建了后台线程，并成功完成了网络连接代码的测试，接下来，我们来深入学习Android线程的知识。

25.4　线程与主线程

网络连接需要时间。Web服务器可能需要1~2秒的时间来响应，文件下载则耗时更久。考虑到这个因素，Android禁止任何主线程网络连接行为。即使强行为之，Android也会抛出`NetworkOnMainThreadException`异常。

这是为什么呢？要想知道答案，首先要知道什么是线程，什么是主线程，主线程有什么用途。

线程是个单一执行序列。单个线程中的代码会逐步执行。所有Android应用的运行都是从主线程开始的。然而，主线程不是线程那样的预定执行序列。相反，它处于一个无限循环的运行状态，等着用户或系统触发事件。一旦有事件触发，主线程便执行代码做出响应。

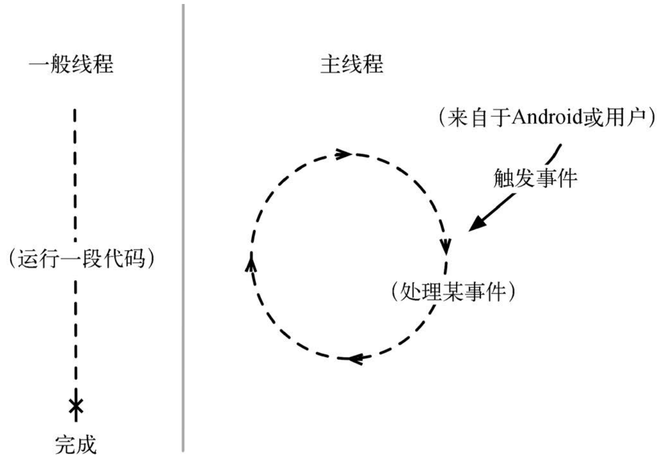


图25-6　一般线程与主线程

把应用想象成一家大型鞋店，闪电侠是这家店唯一的员工。（是不是人人梦寐以求的场景？）要让客户满意，他需要做大量的工作，如布置商品、为顾客取鞋、为顾客量尺寸等。闪电侠并非浪得虚名，所以，即便所有工作都由他一人完成，客户也能得到及时响应，感到满意。

为及时完成任务，闪电侠不能在单一事件上耗时过久。要是一批货丢了怎么办？这时，必须

有人花时间打电话调查此事。假设让闪电侠去做，他在忙于联络查找货物时，店里等候的顾客可就不耐烦了。

闪电侠就像应用里的主线程。它运行着所有更新UI的代码，其中包括响应activity的启动、按钮的点击等不同UI相关事件的代码。（由于响应的事件基本都与用户界面相关，主线程有时也叫作UI线程。）

事件处理循环让UI代码总是按顺序执行。这样，事件就能一件件处理，不用担心互相冲突，同时代码也能够快速执行，及时响应。目前为止，我们编写的所有代码（刚刚使用`AsyncTask`工具类完成的代码除外）都是在主线程中执行的。

超越主线程

连接网络如同致电分销商找丢失的货物：相比其他任务，它更耗时。等待响应期间，用户界面毫无反应，这可能会导致*应用无响应*（application not responding，ANR）现象发生。

如果Android系统监控服务确认主线程无法响应重要事件，如按下后退键等，则应用无响应会发生。用户就会看到如图25-7所示的画面。

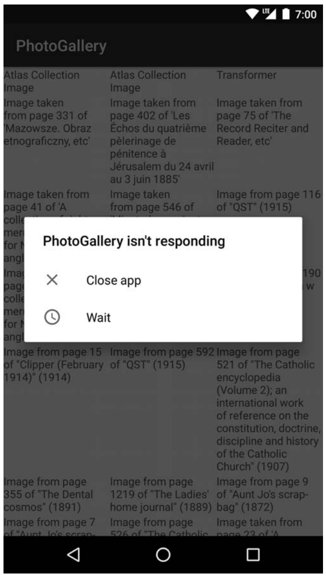


图25-7　应用无响应

回到假想的鞋店中，要解决问题，自然想到再雇一名闪电侠专门负责联络供销商。Android系统中的做法与之类似，即创建一个后台线程，然后从该线程访问网络。

怎样使用后台线程最容易？使用`AsyncTask`工具类。

稍后，还会看到`AsyncTask`类的其他用处。现在，还是先利用网络连接代码做点实事吧。

25.5 从 Flickr 获取 JSON 数据

JSON（JavaScript Object Notation）是近年流行开来的一种数据格式，尤其适用于Web服务。Android提供了标准的org.json包，可以利用包里的一些类创建和解析JSON数据。Android开发者文档有其详细信息。要详细了解JSON数据格式，请访问json.org网站。

Flickr提供了方便而强大的JSON API。可从www.flickr.com/services/api/文档页查看使用细节。在常用浏览器中打开API文档网页，找到Request Formats列表。我们打算使用最简单的REST服务。查文档得知，它的API端点（endpoint）是https://api.flickr.com/services/rest/。可在此端点上调用Flickr提供的方法。

回到API文档主页，找到API Methods列表。向下滚动到photos区域并找到`flickr.photos.getRecent`方法。点击查看该方法。文档对该方法的描述为："返回最近上传到flickr的公共图片清单。"这恰好就是PhotoGallery应用需要的方法。

`getRecent`方法需要的唯一参数是一个API key。为获得它，回到www.flickr.com/ services/api/网页，找到并点击API keys链接进行申请。申请需使用Yahoo ID登录。你可以登录并申请一个非商业用途API key。申请成功后，可获得类似4f721bgafa75bf6d2cb9af54f937bb70这样的API key。（申请API key时，还会得到一个用来访问特定用户信息和图片的Secret key。这里不需要，忽略即可。）

得到API key后，可直接向Flickr网络服务发起一个和下面类似的`GET`请求：

```
https://api.flickr.com/services/rest/?
method=flickr.photos.getRecent&api_key=xxx&format=json&nojsoncallback=1
```

Flickr默认返回XML格式的数据。要获得有效的JSON数据，就需要同时指定`format`和`nojsoncallback`参数。设置`nojsoncallback`为1就是告诉Flickr，返回的数据不应包括封闭方法名和括号。这样才会方便解析数据。

复制上述链接到浏览器，使用刚获取的API key替换xxx字符后回车。很快，就能看到如图25-8所示的JSON返回数据。

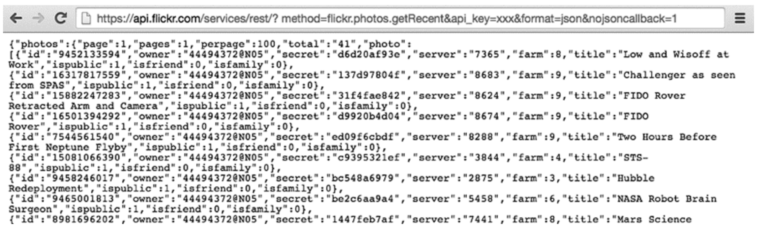


图25-8 JSON数据示例

接下来开始编码。首先，在`FlickrFetchr`类中添加一些常量，如代码清单25-7所示。

代码清单25-7 添加一些常量（FlickrFetchr.java）

```
public class FlickrFetchr {

    private static final String TAG = "FlickrFetchr";
```

```
    private static final String API_KEY = "yourApiKeyHere";
    ...
}
```

记得把yourApiKeyHere替换成刚获取的API key。

使用刚才定义的常量编写一个方法，构建请求URL并获取内容，如代码清单25-8所示。

代码清单25-8　添加fetchItems()方法（FlickrFetchr.java）

```
public class FlickrFetchr {
    ...
    public String getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public void fetchItems() {
        try {
            String url = Uri.parse("https://api.flickr.com/services/rest/")
                    .buildUpon()
                    .appendQueryParameter("method", "flickr.photos.getRecent")
                    .appendQueryParameter("api_key", API_KEY)
                    .appendQueryParameter("format", "json")
                    .appendQueryParameter("nojsoncallback", "1")
                    .appendQueryParameter("extras", "url_s")
                    .build().toString();
            String jsonString = getUrlString(url);
            Log.i(TAG, "Received JSON: " + jsonString);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }
    }
}
```

这里，我们使用Uri.Builder构建了完整的Flickr API请求URL。便利类Uri.Builder可创建正确转义的参数化URL。Uri.Builder.appendQueryParameter(String,String)可自动转义查询字符串。

注意，我们还添加了method、api_key、format和nojsoncallback参数值。另外还指定了一个值为url_s的extras参数。这个参数值告诉Flickr：如有小尺寸图片，也一并返回其URL。

最后，修改PhotoGalleryFragment类中的AsyncTask内部类，调用新的fetchItems()方法，如代码清单25-9所示。

代码清单25-9　调用fetchItems()方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
          @Override
          protected Void doInBackground(Void... params) {
              try {
                  String result = new FlickrFetchr()
                          .getUrlString("https://www.bignerdranch.com");
                  Log.i(TAG, "Fetched contents of URL: " + result);
```

```
                } catch (IOException ioe) {
                    Log.e(TAG, "Failed to fetch URL: ", ioe);
                }
                new FlickrFetchr().fetchItems();
                return null;
            }
        }
    }
```

运行PhotoGallery应用。可看到LogCat窗口中的Flickr JSON数据，如图25-9所示。（在LogCat搜索框中输入FlickrFetchr可以方便查找。）



图25-9 Flickr JSON数据

本书撰写时，Android Studio的LogCat窗口还无法换行显示。想查看JSON字符串完整内容，需向右滚动窗口。（LogCat有时不好伺候。假如看不到类似图25-9的结果，也不用担心。模拟器连接有时不够稳定，可能无法及时显示日志内容。通常，它能自己恢复。实在不行，请重启应用或重启模拟器。）

成功取得Flickr JSON返回结果后，该如何使用呢？和处理其他数据一样，将其存入一个或多个模型对象中。稍后会为PhotoGallery应用创建的模型类名为`GalleryItem`。图25-10为PhotoGallery应用的对象图解。

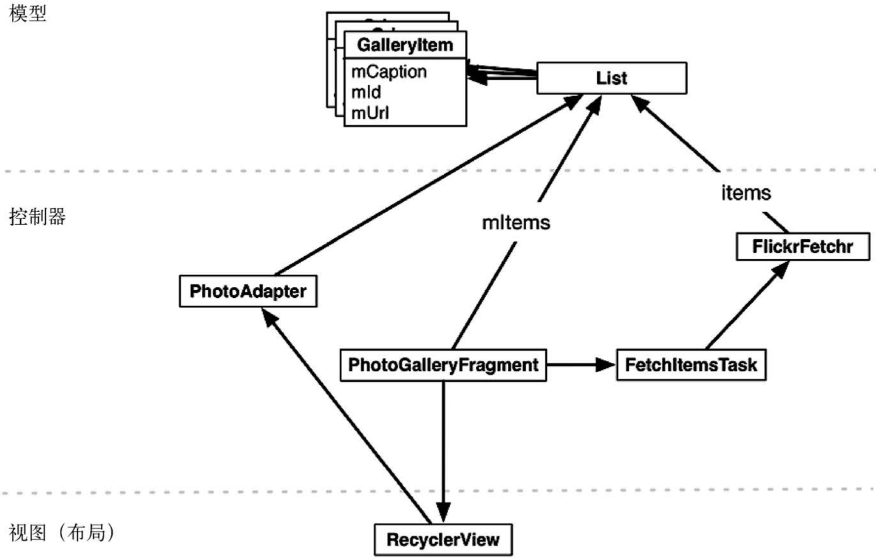


图25-10 PhotoGallery应用的对象图解

注意，为聚焦fragment和网络连接代码，图25-10并没有显示托管activity。

创建`GalleryItem`类并添加有关代码，如代码清单25-10所示。

代码清单25-10　创建模型对象类（GalleryItem.java）

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;

    @Override
    public String toString() {
        return mCaption;
    }
}
```

利用Android Studio自动为`mCaption`、`mId`和`mUrl`变量生成getter与setter方法。

完成模型层对象的创建后，接下来的任务就是塞入JSON解析数据。

解析 JSON 数据

浏览器和LogCat中显示的JSON数据难以阅读。如果用空格回车符格式化后再打印出来，结果大致如图25-11所示。

```
{
  "photos": {
    "page": 1,
    "pages": 10,
    "perpage": 100,
    "total": 1000,
    "photo": [
      {
        "id": "9452133594",
        "owner": "44494372@N05",
        "secret": "d6d20af93e",
        "server": "7365",
        "farm": 8,
        "title": "Low and Wisoff at Work",
        "ispublic": 1,
        "isfriend": 0,
        "isfamily": 0
      }, ...
      {
        "id": "16317817559",
        "owner": "44494372@N05",
        "secret": "137d97804f",
        "server": "8683",
        "farm": 9,
        "title": "Challenger as seen from SPAS",
        "ispublic": 1,
        "isfriend": 0,
        "isfamily": 0
      }
    ]
  },
  "stat": "ok"
}
```

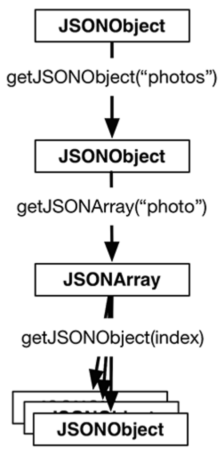


图25-11　格式化后的JSON数据

JSON对象是一系列包含在{ }中的名值对。JSON数组是包含在[ ]中用逗号隔开的JSON对象列表。对象彼此嵌套形成层级关系。

json.org API提供有对应JSON数据的Java对象，如JSONObject和JSONArray。使用JSONObject(String)构造函数，可以很方便地把JSON数据解析进相应的Java对象。更新fetchItems()方法执行解析任务，如代码清单25-11所示。

代码清单25-11　解析JSON数据（FlickrFetchr.java）

```
public class FlickrFetchr {

    private static final String TAG = "FlickrFetchr";
    ...
    public void fetchItems() {
        try {
            ...
            Log.i(TAG, "Received JSON: " + jsonString);
            JSONObject jsonBody = new JSONObject(jsonString);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (JSONException je){
            Log.e(TAG, "Failed to parse JSON", je);
        }
    }
}
```

25

JSONObject构造方法解析传入的Flickr JSON数据后，会生成与原始JSON数据对应的对象树，如图25-11所示。

比较对象树和原始数据可知，顶层JSONObject对应着原始数据最外层的{ }。它包含了一个叫作photos的嵌套JSONObject。层层往下，这个嵌套对象又包含了一个叫作photo的JSONArray。这个嵌套数组中又包含了一组JSONObject，而数组中的一个个JSONObject就是要获取的图片metadata数据。

写一个parseItems(...)方法，取出每张图片的信息，生成一个个GalleryItem对象，再将它们添加到List中，如代码清单25-12所示。

代码清单25-12　解析Flickr图片（FlickrFetchr.java）

```
public class FlickrFetchr {

    private static final String TAG = "FlickrFetchr";
    ...
    public void fetchItems() {
        ...
    }

    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
            throws IOException, JSONException {

        JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
```

```
            JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

            for (int i = 0; i < photoJsonArray.length(); i++) {
                JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

                GalleryItem item = new GalleryItem();
                item.setId(photoJsonObject.getString("id"));
                item.setCaption(photoJsonObject.getString("title"));

                if (!photoJsonObject.has("url_s")) {
                    continue;
                }

                item.setUrl(photoJsonObject.getString("url_s"));
                items.add(item);
            }
        }
    }
```

解析JSONObject层级结构时，上述代码用了getJSONObject(String name)和getJSONArray(String name)这两个便利方法（已标注在图25-11中）。

并不是每张图片都有对应的url_s链接，所以需要添加一个检查。

parseItems(...)方法需要List和JSONObject参数。因此，还要更新fetchItems()方法，让它返回一个包含GalleryItem的List，如代码清单25-13所示。

代码清单25-13　调用parseItems(...)方法（FlickrFetchr.java）

```
    public void List<GalleryItem> fetchItems() {

        List<GalleryItem> items = new ArrayList<>();

        try {
            String url = ...;
            String jsonString = getUrlString(url);
            Log.i(TAG, "Received JSON: " + jsonString);
            JSONObject jsonBody = new JSONObject(jsonString);
            parseItems(items, jsonBody);
        } catch (JSONException je) {
                Log.e(TAG, "Failed to parse JSON", je);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }

        return items;
    }
```

运行PhotoGallery应用，测试JSON解析代码。现在，PhotoGallery应用还无法展示List中的内容。因此，要确认代码是否正确，需设置合适的断点，使用调试器来检查代码逻辑。

25.6 从 AsyncTask 回到主线程

为完成本章的既定目标，我们回到视图层部分，实现在PhotoGalleryFragment类的RecyclerView中显示图片标题。

首先定义一个ViewHolder内部类，如代码清单25-14所示。

代码清单25-14 添加ViewHolder实现（PhotoGalleryFragment.java）

```
        @Override
        public View onCreateView(layoutinflater inflater, ViewGroup container,
                Bundle savedinstanceState) {
            ...
        }

        private class PhotoHolder extends RecyclerView.ViewHolder {
            private TextView mTitleTextView;

            public PhotoHolder(View itemView) {
                super(itemView);

                mTitleTextView = (TextView) itemView;
            }

            public void bindGalleryItem(GalleryItem item) {
                mTitleTextView.setText(item.toString());
            }
        }

        private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
            ...
        }
    }
```

接下来，添加一个RecyclerView.Adapter实现，提供基于GalleryItem对象List的PhotoHolder，如代码清单25-15所示。

代码清单25-15 添加RecyclerView.Adapter实现（PhotoGalleryFragment.java）

```
    public class PhotoGalleryFragment extends Fragment {

        private static final String TAG = "PhotoGalleryFragment";
        ...
        private class PhotoHolder extends RecyclerView.ViewHolder {
            ...
        }

        private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {

            private List<GalleryItem> mGalleryItems;

            public PhotoAdapter(List<GalleryItem> galleryItems) {
                mGalleryItems = galleryItems;
            }
```

25

```
        @Override
        public PhotoHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
            TextView textView = new TextView(getActivity());
            return new PhotoHolder(textView);
        }

        @Override
        public void onBindViewHolder(PhotoHolder photoHolder, int position) {
            GalleryItem galleryItem = mGalleryItems.get(position);
            photoHolder.bindGalleryItem(galleryItem);
        }

        @Override
        public int getItemCount() {
            return mGalleryItems.size();
        }
    }
    ...
}
```

既然RecyclerView要显示的数据已准备就绪，那么接下来编码完成adapter的配置和关联，如代码清单25-16所示。

代码清单25-16　实现setupAdapter()方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    private List<GalleryItem> mItems = new ArrayList<>();
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

        mPhotoRecyclerView = (RecyclerView) v.findViewById(R.id.photo_recycler_view);
        mPhotoRecyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));

        setupAdapter();

        return v;
    }

    private void setupAdapter() {
        if (isAdded()) {
            mPhotoRecyclerView.setAdapter(new PhotoAdapter(mItems));
        }
    }
    ...
}
```

根据当前模型数据（GalleryItem对象List）的状态，刚才添加的setupAdapter()方法会自动配置RecyclerView的adapter。应在onCreateView(...)方法中调用该方法，这样每次因设备旋转重新生成RecyclerView时，可重新为其配置对应的adapter。另外，每次模型层对象发生变化时，也应及时调用该方法。

注意，配置adapter前，应检查isAdded()的返回值是否为true。该检查确认fragment已与目标activity相关联，从而保证getActivity()方法返回结果非空。

还记得吗？fragment可脱离任何activity而独立存在。在这之前，所有的方法调用都是由系统框架的回调方法驱动的，所以不会出现这种情况。本例中，如果fragment在接收回调指令，则它必然关联着某个activity；如它单独存在，也就不会收到回调。

既然在用AsyncTask，说明正在从后台进程触发回调指令。因而不能确定fragment是否关联着activity。那就必须确认fragment是否仍与activity关联。如果没有关联，依赖于activity的操作（如创建PhotoAdapter，进而还会使用托管activity作为context来创建TextView）就会失败。所以，设置adapter之前，你需要确认isAdded()方法返回值。

25

现在，从Flickr成功获取数据后，就需要调用setupAdapter()方法。你的第一反应可能是在FetchItemsTask的doInBackground(...)方法尾部调用它。这不是个好主意。还记得闪电侠与鞋店吗？现在，店里有两个闪电侠，一个忙于应付顾客，一个忙于与Flickr电话沟通。如果第二个闪电侠结束通话后，过来帮忙招呼顾客，会发生什么情况呢？结局很可能是两位闪电侠无法协调一致，产生冲突。

在计算机里，内存对象相互踩踏会让应用崩溃。因此，安全起见，不推荐也不允许从后台线程更新UI。

那么应该怎么做呢？不用担心，AsyncTask还有另一个可覆盖的onPostExecute(...)方法。onPostExecute(...)方法在doInBackground(...)方法执行完毕后才会运行。更为重要的是，它是在主线程而非后台线程上运行的。因此，在该方法中更新UI比较安全。

修改FetchItemsTask类以新的方式更新mItems，并在成功获取图片后调用setupAdapter()方法更新RecyclerView的数据源，如代码清单25-17所示。

代码清单25-17　添加adapter更新代码（PhotoGalleryFragment.java）

```
private class FetchItemsTask extends AsyncTask<Void,Void,~~Void~~ List<GalleryItem>> {
    @Override
    protected ~~Void~~ List<GalleryItem> doInBackground(Void... params) {

        return new FlickrFetchr().fetchItems();
        ~~return null;~~
    }

    @Override
    protected void onPostExecute(List<GalleryItem> items) {
        mItems = items;
        setupAdapter();
    }
}
```

上述代码有三处调整。首先，我们改变了FetchItemsTask类第三个泛型参数的类型。该参数是AsyncTask返回结果的数据类型。也就是doInBackground(...)方法返回结果的数据类型，以及onPostExecute(...)方法输入参数的数据类型。

其次，我们让doInBackground(...)方法返回了GalleryItem对象List。这样既修正了代码编译错误，还将GalleryItem对象List传递给了onPostExecute(...)方法。

最后，我们添加了onPostExecute(...)方法实现代码。该方法接收doInBackground(...)方法返回的GalleryItem数据，并放入mItems变量，然后调用setupAdapter()方法更新RecyclerView视图的adapter。

至此，本章任务就完成了。运行PhotoGallery应用，可看到屏幕上显示出全部已下载GalleryItem的标题（类似图25-2）。

25.7　清理 AsyncTask

本章，AsyncTask运用得还算得当，因此不用去管理AsyncTask实例了。例如，我们保留了fragment（调用setRetainInstance(true)方法），这样即使设备旋转，也不会重复创建新的AsyncTask去获取JSON数据。然而，有些情况下，必须好好掌控它，必要时，甚至要能撤销或重新运行AsyncTask。

针对某些复杂应用场景，我们需要将AsyncTask赋值给实例变量。这样，一旦掌控了它，就能随时调用AsyncTask.cancel(boolean)方法，撤销运行中的AsyncTask。

AsyncTask.cancel(boolean)方法有两种工作模式：粗暴的和温和的。如果调用cancel(false)方法，它只是简单地设置isCancelled()的状态为true。随后，AsyncTask会检查isCancelled()状态，然后选择提前结束运行。

然而，如果调用cancel(true)方法，它会立即终止doInBackground(...)方法当前所在的线程。AsyncTask.cancel(true)方法停止AsyncTask的方式简单粗暴，如果可能，应尽量避免。

应该在什么时候、什么地方撤销AsyncTask呢？这要看情况了。先问问自己，如果fragment或activity已销毁了或是看不到了，AsyncTask当前的工作可以停止吗？如果可以，就在onStop(...)方法里（看不到视图），或者在onDestroy(...)方法里（fragment/activity实例已销毁）撤销AsyncTask实例。

即使fragment/activity已销毁了（或者视图已看不到了），也可以不撤销AsyncTask，让它运行至结束把事情做完。不过，这可能会引发内存泄漏（比如，没用的Activity实例本应销毁，但一直还在内存里），也可能会出现UI更新问题（因为UI已失效）。如果不管用户怎么操作，要确保重要工作能完成，那最好考虑其他解决方案，比如使用Service（详见第28章）。

25.8　深入学习：AsyncTask 再探

你已知道如何使用AsyncTask的第三个类型参数，那另外两个类型参数呢？

第一个类型参数可指定将要转给execute(...)方法的输入参数的类型，进而确定

doInBackground(...)方法输入参数的类型。具体用法可参考以下示例：

```
AsyncTask<String,Void,Void> task = new AsyncTask<String,Void,Void>() {
    public Void doInBackground(String... params) {
        for (String parameter : params) {
            Log.i(TAG, "Received parameter: " + parameter);
        }

        return null;
    }
};
```

输入参数传入execute(...)方法（可接受一个或多个参数）：

```
task.execute("First parameter", "Second parameter", "Etc.");
```

然后，再把这些变量参数传递给doInBackground(...)方法。

第二个类型参数可指定发送进度更新需要的类型。以下为示例代码：

```
final ProgressBar gestationProgressBar = /* A determinate progress bar */;
gestationProgressBar.setMax(42); /* max allowed gestation period */

AsyncTask<Void,Integer,Void> haveABaby = new AsyncTask<Void,Integer,Void>() {
    public Void doInBackground(Void... params) {
        while (!babyIsBorn()) {
            Integer weeksPassed = getNumberOfWeeksPassed();
            publishProgress(weeksPassed);
            patientlyWaitForBaby();
        }
    }

    public void onProgressUpdate(Integer... params) {
        int progress = params[0];
        gestationProgressBar.setProgress(progress);
    }
};

/* call when you want to execute the AsyncTask */
haveABaby.execute();
```

进度更新通常发生在后台进程执行中途。问题是，在后台进程中无法完成必要的UI更新。因此AsyncTask提供了publishProgress(...)和onProgressUpdate(...)方法。

其工作方式是这样的：在后台线程中，从doInBackground(...)方法中调用publishProgress(...)方法。这样onProgressUpdate(...)方法便能够在UI线程上调用。因此，在onProgressUpdate(...)方法中执行UI更新就可行了，但必须在doInBackground(...)方法中使用publishProgress(...)方法对它们进行管控。

25.9 深入学习：AsyncTask 的替代方案

在使用AsyncTask加载数据时，如果遇到设备配置变化，比如设备旋转，你得负责管理它的

生命周期，同时还要保存好数据，不让其因旋转丢失。虽然调用Fragment的setRetainInstance(true)方法来保存数据可以解决问题，但它不是万能的。很多时候，你还得介入，编写特殊场景应对代码，让应用无懈可击。这些特殊场景有：用户在AsyncTask运行时按后退键，以及启动AsyncTask的fragment因内存紧张而被销毁。

使用Loader是另一种可行的解决方案。它可以代劳很多（并非全部）棘手的事情。Loader用来从某些数据源加载数据（对象）。数据源可以是磁盘、数据库、ContentProvider、网络，甚至是另一进程。

AsyncTaskLoader是个抽象Loader。它可以使用AsyncTask把数据加载工作转移到其他线程上。我们创建的loader类几乎都是AsyncTaskLoader的子类。AsyncTaskLoader能在不阻塞主线程的前提下获取到数据，并把结果发送给目标对象。

相比AsyncTask，为什么要推荐使用loader呢？最重要的原因是，遇到类似设备旋转这样的场景时，LoaderManager会帮我们妥善管理loader及其加载的数据。而且，LoaderManager还负责启动和停止loader，以及管理loader的生命周期。怎么样？理由充足吧！

设备配置改变后，如果初始化一个已经加载完数据的loader，它能立即提交数据，而不是再次尝试获取数据。无论fragment是否得到保留，它都会这样做。这下放心多了，从此再也不用考虑因保留fragment而产生的生命周期问题了。

25.10　挑战练习：Gson

无论什么平台，把JSON数据转化为Java对象都是应用开发的常见任务，如代码清单25-12所做的那样。于是，聪明的开发者就创建了一些工具库，希望能简化JSON数据和Java对象的互转。

Gson就是这样的一个工具库（github.com/google/gson）。不用写任何解析代码，Gson就能自动把JSON数据映射为Java对象。因为这个特性，Gson现在是开发者最喜爱的JSON解析库。

挑战自己，在应用中整合Gson库，简化FlickrFetchr中的JSON解析。

25.11　挑战练习：分页

getRecent方法默认返回一页包含100个结果的数据。不过，该方法还有个叫作page的参数，可以用它返回第二页、第三页等更多页数据。

请实现一个RecyclerView.OnScrollListener方法，只要用户看完当前页，就使用下页返回结果替换当前页。想更有挑战的话，可以尝试把后续结果页添加到当前结果页后面。

25.12　挑战练习：动态调整网格列

当前，显示图片标题的网格固定有3列。编写代码动态调整网格列数，实现在横屏或大屏幕设备上显示更多列标题。

实现这个目标有个简单方法：分别为不同的设备配置或屏幕尺寸提供整数修饰资源。这实际

和第17章中为不同尺寸屏幕提供不同布局的方式差不多。整数修饰资源应放置在res/values目录中。具体实施细节可参阅Android开发者文档。

提供整数修饰资源的方式不太好确定网格列细分粒度（只能凭经验预先定义列数）。下面再介绍一个颇具挑战的方法：在fragment的视图创建时就计算并设置好网格列数。显然，这种方式更加灵活实用。基于`RecyclerView`的当前宽度和预定义网格列宽，就可以计算出列数。

实施前还有个问题要解决：你不能在`onCreateView()`方法中计算网格列数，因为这个时候`RecyclerView`还没有改变。不过，可以实现`ViewTreeObserver.OnGlobalLayoutListener`监听器方法和计算列数的`onGlobalLayout()`方法，然后使用`addOnGlobalLayoutListener()`把监听器添加给`RecyclerView`视图。

第 26 章 Looper、Handler和HandlerThread

从Flickr下载并解析JSON数据后，接下来的任务就是下载并显示图片。本章，我们来学习如何使用Looper、Handler和HandlerThread动态下载和显示图片。

26.1　配置 RecyclerView 以显示图片

在PhotoGalleryFragment中，当前PhotoHolder准备了TextView供RecyclerView的GridLayoutManager显示。每个TextView显示一个GalleryItem标题。

要显示图片，就要让PhotoHolder提供ImageView。最终，每个ImageView都应显示一张从GalleryItem的mUrl地址下载的图片。

首先，为GalleryItem创建一个名为list_item_gallery.xml的布局文件。该布局包含一个ImageView组件，如图26-1所示。

```
ImageView
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/item_image_view"
android:layout_width="match_parent"
android:layout_height="120dp"
android:layout_gravity="center"
android:scaleType="centerCrop"
```

图26-1　Gallery图片项布局（res/layout/list_item_gallery.xml）

ImageView由RecyclerView的GridLayoutManager负责管理，这意味着其宽度会变，而高度固定。为最大化利用ImageView的空间，应设置它的scaleType属性值为centerCrop。这个属性值的作用是先居中放置图片，然后放大较小图片，裁剪较大图片（裁两头）以匹配视图。

接下来更新PhotoHolder类，替换掉TextView，让其保存ImageView。同时，用一个新方法替换掉bindGalleryItem()方法，用来设置ImageView的Drawable，如代码清单26-1所示。

代码清单26-1　更新PhotoHolder（PhotoGalleryFragment.java）

```
private class PhotoHolder extends RecyclerView.ViewHolder {
    private TextView mTitleTextView ImageView mItemImageView;

    public PhotoHolder(View itemView) {
        super(itemView);

        mTitleTextView = (TextView) itemView;
        mItemImageView = (ImageView) itemView.findViewById(R.id.item_image_view);
    }

    public void bindGalleryItem(GalleryItem item) {
        mTitleTextView.setText(item.toString());
    }

    public void bindDrawable(Drawable drawable) {
        mItemImageView.setImageDrawable(drawable);
    }
}
```

之前，传入PhotoHolder构造方法的是TextView。现在，新版本PhotoHolder构造方法需要的是一个资源ID为R.id.item_image_view的ImageView。

更新PhotoAdapter的onCreateViewHolder(...)方法，实例化list_item_gallery布局。然后将结果返回给PhotoAdapter的构造方法，如代码清单26-2所示。

代码清单26-2　更新PhotoAdapter的onCreateViewHolder()方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
        ...
        @Override
        public PhotoHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
            TextView textView = new TextView(getActivity());
            return new PhotoHolder(textView);
            LayoutInflater inflater = LayoutInflater.from(getActivity());
            View view = inflater.inflate(R.layout.list_item_gallery, viewGroup, false);
            return new PhotoHolder(view);
        }
        ...
    }
  ...
}
```

现在，需要为每个ImageView设置占位图，等成功下载图片后再做替换。在随书代码文件中找到bill_up_close.png，把它复制到项目的res/drawable目录中。

更新PhotoAdapter的onBindViewHolder()方法，使用占位图设置ImageView的Drawable，如代码清单26-3所示。

代码清单26-3　绑定默认图片（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
```

```
    ...
    private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
        ...
        @Override
        public void onBindViewHolder(PhotoHolder photoHolder, int position) {
          GalleryItem galleryItem = mGalleryItems.get(position);
          photoHolder.bindGalleryItem(galleryItem);
          Drawable placeholder = getResources().getDrawable(R.drawable.bill_up_close);
          photoHolder.bindDrawable(placeholder);
        }
        ...
    }
    ...
}
```

运行PhotoGallery应用，Bill的一组大头照出现了，如图26-2所示。



图26-2　满屏的Bill大头照

26.2　批量下载缩略图

当前，PhotoGallery应用联网代码的工作方式如下：PhotoGalleryFragment执行一个AsyncTask，该AsyncTask在后台线程上从Flickr获取JSON数据，然后解析JSON并将解析结果存入GalleryItem数组。最终每个GalleryItem都得到一个指向某张缩略图的URL。

接下来是下载这些URL指向的缩略图。是不是认为只要在FetchItemsTask的doInBackground()方法中添加一些网络下载代码就行了？GalleryItem数组含有100个URL下载链接。我们每次下载一张，直到下完100张。最后，执行onPostExecute(...)方法，让所有下载的图片全部显示在RecyclerView视图中。

然而，一次性下载全部缩略图存在两个问题。首先，下载比较耗时，而且在下载完成前，UI都无法完成更新。这样，网速较慢时，用户就只能对着Bill的照片看好久。

其次，保存缩略图也是个问题。100张缩略图保存在内存中固然轻松，但是1000张呢？如果还需要实现无限滚动来显示图片呢？显然，内存会耗尽。

考虑到这类问题，很多应用通常会选择仅在需要显示图片时才去下载。显然，RecyclerView及其adapter应负责实现按需下载。adapter触发图片下载就放在onBindViewHolder(...)方法中实现。

AsyncTask是执行后台线程的最简单方式，但它不适用于那些重复且长时间运行的任务。（具体原因，请阅读章末深入学习部分的内容。）

放弃AsyncTask吧，我们来创建一个专用的后台线程。这是实现按需下载的最常用方式。

26.3 与主线程通信

虽然我们打算让专用线程负责下载图片，但在无法与主线程直接通信的情况下，它是如何协同RecyclerView的adapter来实现图片显示的呢？

再次回到闪电侠与鞋店的假想场景。后台工作的闪电侠已结束与分销商的电话沟通。他需要将库存已找回的消息通知给前台闪电侠。如果前台闪电侠非常忙碌，后台闪电侠就不能打扰他。于是，他选择登记预约，等前台闪电侠空闲下来再联系。这虽然可行，但效率不高。

比较好的解决方案是为每个闪电侠提供一个收件箱。后台闪电侠写下鞋已入库的信息，并将其放置在前台闪电侠的收件箱顶部。前台闪电侠如需告诉后台闪电侠库存已空的信息，也可以这样做。

实践证明，收件箱的办法非常好用。有时，闪电侠可能需要尽快完成一项任务，但不方便立即去做。这种情况下，他也可以在自己的收件箱放上一条提醒消息，等有空了就赶紧去处理。

Android系统中，线程使用的收件箱叫作*消息队列*（message queue）。使用消息队列的线程叫作*消息循环*（message loop）。消息循环会循环检查队列上是否有新消息，如图26-3所示。

消息循环由线程和looper组成。Looper对象管理着线程的消息队列。

主线程就是个消息循环，因此也拥有looper。主线程的所有工作都是由其looper完成的。looper不断从消息队列中抓取消息，然后完成消息指定的任务。

接下来，我们将创建一个消息循环作为后台线程。准备需要的looper时，我们会使用名为HandlerThread的类。

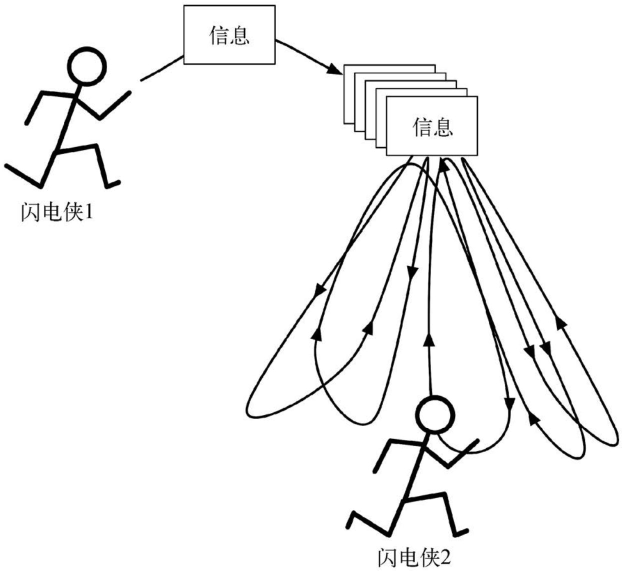


图26-3　闪电侠之舞

26.4　创建并启动后台线程

继承HandlerThread类，创建一个名为ThumbnailDownloader的新类。然后，添加一个构造方法，一个名为queueThumbnail()的存根方法以及一个quit()覆盖方法（线程退出通知方法，稍后会用到），如代码清单26-4所示。

代码清单26-4　初始线程代码（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";

    private Boolean mHasQuit = false;

    public ThumbnailDownloader() {
        super(TAG);
    }

    @Overide
    public boolean quit() {
        mHasQuit = true;
        return super.quit();
    }
    public void queueThumbnail(T target, String url) {
        Log.i(TAG, "Got a URL: " + url);
    }
}
```

注意，ThumbnailDownloader类使用了<T>泛型参数。ThumbnailDownloader类的使用者（这里指PhotoGalleryFragment），需要使用某些对象来识别每次下载，并确定该使用已下载图片更新哪个UI元素。有了泛型参数，实施起来方便了很多。

queueThumbnail()方法需要一个T类型对象（标识具体哪次下载）和一个String参数（URL下载链接）。同时，它也是PhotoAdapter在其onBindViewHolder(...)实现方法中要调用的方法。

打开PhotoGalleryFragment.java文件，为PhotoGalleryFragment添加一个Thumbnail-Downloader类型的成员变量。然后，在onCreate(...)方法中，创建并启动线程。最后，覆盖onDestroy()方法退出线程，如代码清单26-5所示。

代码清单26-5 创建ThumbnailDownloader（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    private List<GalleryItem> mItems = new ArrayList<>();
    private ThumbnailDownloader<PhotoHolder> mThumbnailDownloader;
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        new FetchItemsTask().execute();

        mThumbnailDownloader = new ThumbnailDownloader<>();
        mThumbnailDownloader.start();
        mThumbnailDownloader.getLooper();
        Log.i(TAG, "Background thread started");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mThumbnailDownloader.quit();
        Log.i(TAG, "Background thread destroyed");
    }
    ...
}
```

ThumbnailDownloader的泛型参数支持任何对象，但在这里，PhotoHolder最合适，因为该视图是最终显示下载图片的地方。

上述代码有两点安全考虑，值得一说。

- ThumbnailDownloader的getLooper()方法是在start()方法之后调用的。(稍后会学习更多有关Looper的知识。) 这能保证线程就绪，避免潜在竞争（尽管极少发生）。因为getLooper()方法能执行成功，说明onLooperPrepared()方法肯定早已完成。这样，queueThumbnail()方法因Handler为空而调用失败的情况就能避免了。
- 在onDestroy()方法内调用quit()方法结束线程。这非常关键。如不终止Handler-Thread，它会一直运行下去，成为僵尸。

最后，在PhotoAdapter.onBindViewHolder(...)方法中，调用线程的queueThumbnail()方法，并传入放置图片的PhotoHolder和GalleryItem的URL，如代码清单26-6所示。

代码清单26-6　让ThumbnailDownloader跑起来（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
  ...
  private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
    ...
    @Override
    public void onBindViewHolder(PhotoHolder photoHolder, int position) {
        GalleryItem galleryItem = mGalleryItems.get(position);
        Drawable placeholder = getResources().getDrawable(R.drawable.bill_up_close);
        photoHolder.bindDrawable(placeholder);
        mThumbnailDownloader.queueThumbnail(photoHolder, galleryItem.getUrl());
    }
    ...
  }
    ...
}
```

运行PhotoGallery应用并查看LogCat窗口。在RecyclerView视图中滑动时，可看到ThumbnailDownloader正在后台处理各个下载请求。

成功创建并运行HandlerThread线程后，接下来的任务是：使用传入queueThumbnail()方法的信息创建消息，并放置在ThumbnailDownloader的消息队列中。

26.5　Message 与 message handler

创建消息前，首先要理解什么是Message，以及它与Handler（或者说message handler）之间的关系。

26.5.1　剖析 Message

首先来看消息。闪电侠收件箱里的消息是需要处理的各种任务消息。“你跑得真快!”这样的鼓励消息是没空写的。

消息是Message类的一个实例，它有好几个实例变量，其中有三个需要你定义。

- What：用户定义的int型消息代码，用来描述消息。
- obj：用户指定，随消息发送的对象。

- target：处理消息的Handler。

Message的目标（target）是一个Handler类实例。Handler可看作message handler的简称。创建Message时，它会自动与一个Handler相关联。Message待处理时，Handler对象负责触发消息处理事件。

26.5.2　剖析 Handler

要处理消息以及消息指定的任务，首先需要一个Handler实例。Handler不仅仅是处理Message的目标（target），也是创建和发布Message的接口，如图26-4所示。

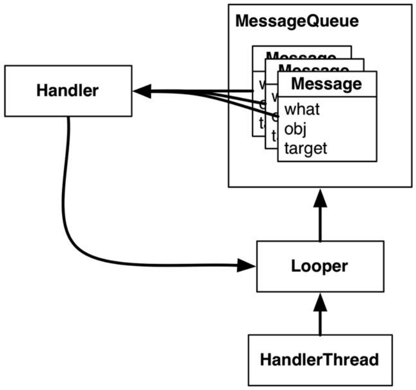


图26-4　Looper、Handler、HandlerThread和Message

Looper拥有Message对象的收件箱，所以Message必须在Looper上发布或处理。既然有这层关系，为协同工作，Handler总是引用着Looper。

一个Handler仅与一个Looper相关联，一个Message也仅与一个目标Handler（也称作Message目标）相关联，如图26-5所示。Looper拥有整个Message队列。多个Message可以引用同一目标Handler。

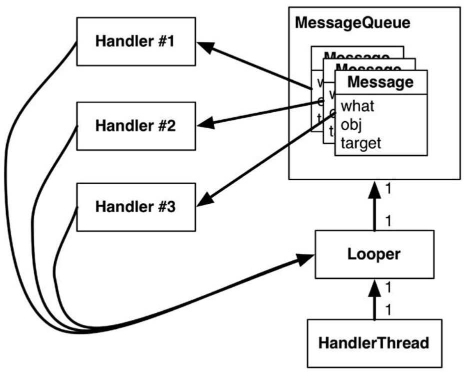


图26-5　多个Handler对应一个Looper

多个Handler也可与一个Looper相关联。这意味着一个Handler的Message可能与另一个Handler的Message存放在同一消息队列中。

26.5.3　使用 handler

一般来讲，不应手动设置消息的目标Handler。创建信息时，最好调用Handler.obtainMessage (...)方法。传入其他必要消息字段后，该方法会自动设置目标Handler。

为避免反复创建新的Message对象，Handler.obtainMessage(...)方法会从公共回收池里获取消息。相比创建新实例，这样更加高效。

一旦取得Message，就可以调用sendToTarget()方法将其发送给它的Handler。然后，Handler会将这个Message放置在Looper消息队列的尾部。

对于PhotoGallery应用，我们会在queueThumbnail()实现方法中获取并发送消息给它的目标。消息的what属性是一个定义为MESSAGE_DOWNLOAD的常量。消息的obj属性是一个T类型对象，这里指由adapter传入queueThumbnail()方法的PhotoHolder，用于标识下载。

Looper取得消息队列中的特定消息后，会将它发送给消息的目标Handler去处理。消息一般是在目标Handler的Handler.handleMessage(...)实现方法中进行处理的。

图26-6展示了其中的对象关系。

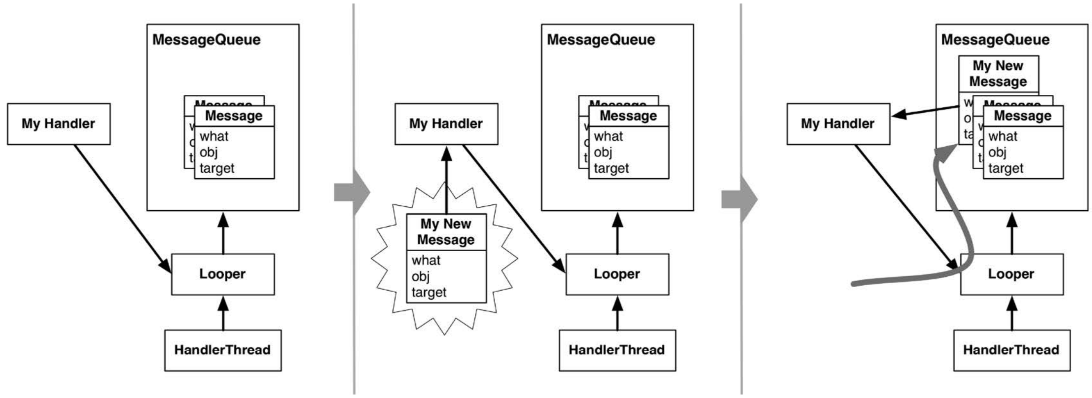


图26-6　创建并发送Message

这里，稍后要创建的handleMessage(...)实现方法将使用FlickrFetchr从URL下载图片字节数据，然后再转换为位图。

开始写代码，首先添加一些常量和成员变量，如代码清单26-7所示。

代码清单26-7　添加一些常量和成员变量（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;
```

```
    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();
    ...
}
```

MESSAGE_DOWNLOAD用来标识下载请求消息。（ThumbnailDownloader会把它设为任何新创建下载消息的what属性。）

新添加的mRequestHandler用来存储对Handler的引用。这个Handler负责在ThumbnailDownloader后台线程上管理下载请求消息队列。还负责从消息队列里取出并处理下载请求消息。

新添加的mRequestMap是个ConcurrentHashMap。这是一种线程安全的HashMap。这里，使用一个标记下载请求的T类型对象作为key，我们可以存取和请求关联的URL下载链接。（这个标记对象是PhotoHolder，下载结果就能很方便地发送给显示图片的UI元素。）

接下来，在queueThumbnail(...)方法中添加代码，更新mRequestMap并把下载消息放到后台线程的消息队列中去，如代码清单26-8所示。

代码清单26-8　获取和发送消息（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();

    public ThumbnailDownloader() {
        super(TAG);
    }

    @Override
    public boolean quit() {
        mHasQuit = true;
        return super.quit();
    }

    public void queueThumbnail(T target, String url) {
        Log.i(TAG, "Got a URL: " + url);

        if (url == null) {
            mRequestMap.remove(target);
        } else {
            mRequestMap.put(target, url);
            mRequestHandler.obtainMessage(MESSAGE_DOWNLOAD, target)
                    .sendToTarget();
        }
    }
}
```

从mRequestHandler直接获取到消息后，mRequestHandler也就自动成为了这个新Message

对象的target。这表明mRequestHandler会负责处理从消息队列中取出的这个消息。这个消息的what属性是MESSAGE_DOWNLOAD。它的obj属性是传递给queueThumbnail(...)方法的T target值（这里指PhotoHolder）。

新消息就代表指定为T target（RecyclerView中的PhotoHolder）的下载请求。还记得吗？PhotoGalleryFragment中，RecyclerView的adapter就是从onBindViewHolder(...)方法里调用queueThumbnail(...)，把待下载图片及其URL传给PhotoHolder的。

注意，消息自身不包含URL信息。我们的做法是使用PhotoHolder和URL的对应关系更新mRequestMap。随后，我们会从mRequestMap中取出图片URL，以保证总是使用了匹配PhotoHolder实例的最新下载请求URL。（这很重要，因为RecyclerView中的ViewHolder是会不断回收重用的。）

最后，初始化mRequestHandler并定义该Handler在得到消息队列中的下载消息后应执行的任务，如代码清单26-9所示。

代码清单26-9　处理消息（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();

    public ThumbnailDownloader() {
        super(TAG);
    }

    @Override
    protected void onLooperPrepared() {
        mRequestHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                if (msg.what == MESSAGE_DOWNLOAD) {
                    T target = (T) msg.obj;
                    Log.i(TAG, "Got a request for URL: " + mRequestMap.get(target));
                    handleRequest(target);
                }
            }
        };
    }

    @Override
    Public boolean quit(){
        mHasQuit = true;
        return super.quit();
    }

    public void queueThumbnail(T target, String url) {
        ...
```

```
    }

    private void handleRequest(final T target) {
        try {
            final String url = mRequestMap.get(target);

            if (url == null) {
                return;
            }

            byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
            final Bitmap bitmap = BitmapFactory
                    .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
            Log.i(TAG, "Bitmap created");

        } catch (IOException ioe) {
            Log.e(TAG, "Error downloading image", ioe);
        }
    }
}
```


上述代码中，我们是在onLooperPrepared()方法里实现Handler.handleMessage(...)方法的。HandlerThread.onLooperPrepared()是在Looper首次检查消息队列之前调用，所以该方法是创建Handler实现的好地方。

在Handler.handleMessage(...)方法中，首先检查消息类型，再获取obj值（T类型下载请求），然后将其传递给handleRequest(...)方法处理。（前面说过，队列中的下载消息取出并可以处理时，就会触发调用Handler.handleMessage(...)方法。）

handleRequest()方法是下载执行的地方。在这里，确认URL有效后，就将它传递给FlickrFetchr新实例。确切地说，此处使用的是上一章中创建的FlickrFetchr.getUrlBytes(...)方法。

最后，使用BitmapFactory把getUrlBytes(...)返回的字节数组转换为位图。

运行PhotoGallery应用，通过LogCat窗口的日志确认代码工作正常。

当然，在将位图设置给PhotoHolder视图（来自于PhotoAdapter）之前，请求处理还不算完。不过这是UI的工作。因此，必须回到主线程上完成它。

目前为止，所有的工作就是在线程上使用handler和消息——ThumbnailDownloader把消息放入自己的收件箱。下一节要学习的内容是：ThumbnailDownloader如何使用Handler向主线程发请求。

26.5.4 传递 handler

当前，使用ThumbnailDownloader的mRequestHandler，我们已可以从主线程安排后台线程任务，如图26-7所示。

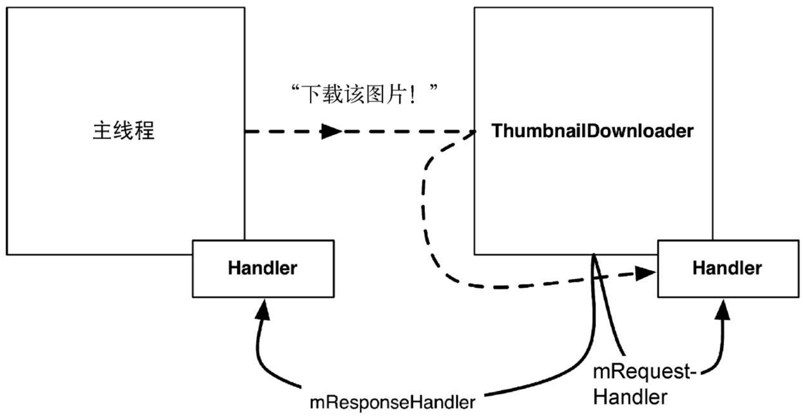


图26-7　从主线程安排ThumbnailDownloader上的任务

反过来，也可以从后台线程使用与主线程关联的Handler，安排主线程任务，如图26-8所示。

主线程是一个拥有handler和Looper的消息循环。主线程上创建的Handler会自动与它的Looper相关联。主线程上创建的这个Handler也可以传递给另一线程。传递出去的Handler与创建它的线程Looper始终保持着联系。因此，已传出Handler负责处理的所有消息都将在主线程的消息队列中处理。

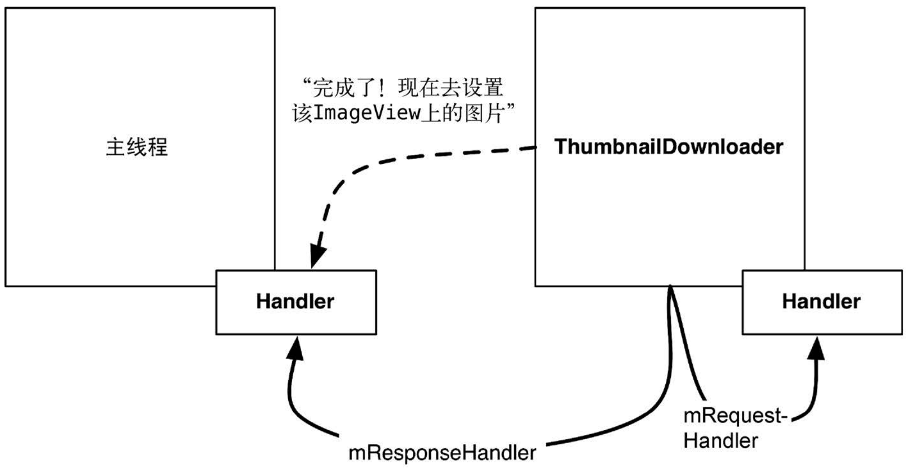


图26-8　从ThumbnailDownloader线程上规划主线程任务

在ThumbnailDownloader.java中，添加上图中的mResponseHandler变量，以存放来自于主线程的Handler。然后，以一个能接受Handler的构造方法替换原构造方法，并设置变量的值。最后新增一个监听器接口响应请求（主线程发请求，响应结果是下载的图片），如代码清单26-10所示。

代码清单26-10 处理消息（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentMap<T,String> mRequestMap = new ConcurrentHashMap<>();
    private Handler mResponseHandler;
    private ThumbnailDownloadListener<T> mThumbnailDownloadListener;

    public interface ThumbnailDownloadListener<T> {
        void onThumbnailDownloaded(T target, Bitmap thumbnail);
    }

    public void setThumbnailDownloadListener(ThumbnailDownloadListener<T> listener) {
        mThumbnailDownloadListener = listener;
    }

    public ThumbnailDownloader(Handler responseHandler) {
        super(TAG);
        mResponseHandler = responseHandler;
    }
    ...
}
```


在图片下载完成，可以交给UI去显示时，定义在ThumbnailDownloadListener新接口中的onThumbnailDownloaded(...)方法就会被调用。稍后，为了把下载任务和UI更新任务（把图片放入ImageView）分开，代替ThumbnailDownloader，我们会使用这个监听器方法把处理已下载图片的任务委托给另一个类（这里指PhotoGalleryFragment）。这样，ThumbnailDownloader就可以把下载结果传给其他视图对象。

接下来，修改PhotoGalleryFragment类，将主线程关联的Handler传递给ThumbnailDownloader。另外，再设置一个ThumbnailDownloadListener处理已下载图片，如代码清单26-11所示。

代码清单26-11 使用反馈Handler（PhotoGalleryFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    new FetchItemsTask().execute();

    Handler responseHandler = new Handler();
    mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
    mThumbnailDownloader.setThumbnailDownloadListener(
        new ThumbnailDownloader.ThumbnailDownloadListener<PhotoHolder>() {
            @Override
            public void onThumbnailDownloaded(PhotoHolder photoHolder, Bitmap bitmap) {
                Drawable drawable = new BitmapDrawable(getResources(), bitmap);
                photoHolder.bindDrawable(drawable);
            }
        }
```

```
        );
        mThumbnailDownloader.start();
        mThumbnailDownloader.getLooper();
        Log.i(TAG, "Background thread started");
    }
```

前面说过，Handler默认与当前线程的Looper相关联。这个Handler是在onCreate(...)方法中创建的，所以它会与主线程的Looper相关联。

现在，通过mResponseHandler，ThumbnailDownloader能够使用与主线程Looper绑定的Handler。同时，还有ThumbnailDownloadListener使用返回的Bitmap执行UI更新操作。具体来说，就是通过onThumbnailDownloaded实现，使用新下载的Bitmap来设置PhotoHolder的Drawable。

和在后台线程上把图片下载请求放入消息队列类似，我们也可以发送定制Message给主线程，要求显示已下载图片。不过，这需要另一个Handler子类，以及一个handleMessage(...)覆盖方法。

方便起见，我们转而使用另一个Handler方法——post(Runnable)。

Handler.post(Runnable)是一个发布Message的便利方法。示例如下：

```
Runnable myRunnable = new Runnable() {
    @Override
    public void run() {
        /* Your code here */
    }
};
Message m = mHandler.obtainMessage();
m.callback = myRunnable;
```

Message设有回调方法属性后，取出队列的消息是不会发给target Handler的。相反，存储在回调方法中的Runnable的run()方法会直接执行。

在ThumbnailDownloader.handleRequest()方法中，添加代码清单26-12所示代码。

代码清单26-12　图片下载与显示（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    ...
    private Handler mResponseHandler;
    private ThumbnailDownloadListener<T> mThumbnailDownloadListener;
    ...
    private void handleRequest(final T target) {
        try {
            final String url = mRequestMap.get(target);

            if (url == null) {
                return;
            }

            byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
            final Bitmap bitmap = BitmapFactory
                    .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
            Log.i(TAG, "Bitmap created");
```

```
                mResponseHandler.post(new Runnable() {
                    public void run() {
                        if (mRequestMap.get(target) != url ||
                                mHasQuit) {
                            return;
                        }

                        mRequestMap.remove(target);
                        mThumbnailDownloadListener.onThumbnailDownloaded(target, bitmap);
                    }
                });

            } catch (IOException ioe) {
                Log.e(TAG, "Error downloading image", ioe);
            }
        }
    }
```

因为mResponseHandler与主线程的Looper相关联，所以UI更新代码会在主线程中完成。

那么上述代码有什么作用呢？首先，它再次检查requestMap。这很有必要，因为RecyclerView会循环使用其视图。在ThumbnailDownloader下载完成Bitmap之后，RecyclerView可能循环使用了PhotoHolder并相应请求了一个不同的URL。该检查可保证每个PhotoHolder都能获取到正确的图片，即使中间发生了其他请求也无妨。

接下来，检查mHasQuit值。如果ThumbnailDownloader已经退出，运行任何回调方法可能都不太安全。

最后，从requestMap中删除配对的PhotoHolder-URL，然后将位图设置到目标PhotoHolder上。

在运行应用并欣赏图片前，还应考虑一个风险点。如果用户旋转屏幕，因PhotoHolder视图的失效，ThumbnailDownloader可能会挂起。如果点击这些ImageView，就会发生异常。

新增clearQueue()方法清除队列中的所有请求，如代码清单26-13所示。

代码清单26-13　添加清理方法（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    ...
    public void queueThumbnail(T target, String url) {
        ...
    }

    public void clearQueue() {
        mRequestHandler.removeMessages(MESSAGE_DOWNLOAD);
        mRequestMap.clear();
    }

    private void handleRequest(final T target) {
        ...
    }
}
```

既然视图已销毁，别忘了在PhotoGalleryFragment中清空mThumbnailDownloader，如代码清单26-14所示。

代码清单26-14　调用清理方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
  ...
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
      ...
  }

  @Override
  public void onDestroyView() {
      super.onDestroyView();
      mThumbnailDownloader.clearQueue();
  }

  @Override
  public void onDestroy() {
      ...
  }
  ...
}
```

至此，本章的所有任务都完成了。运行PhotoGallery应用，滚动屏幕查看图片的动态加载。

PhotoGallery应用有了下载并显示图片的基本功能。接下来的几章还会为应用添加更多功能，如搜索图片、在Web视图中打开图片所在的Flickr网页等。

26.6　深入学习：AsyncTask 与线程

理解了Handler和Looper之后，AsyncTask也就没有当初看上去那么神奇了。不过就本章所做的线程相关工作来看，要用AsyncTask能省不少事。那么为什么要用HandlerThread，而不用它呢?

原因有好几个。最基本的一个是AsyncTask的工作方式并不适用于本章的使用场景。它主要应用于那些短暂且较少重复的任务。上一章的应用场景才是AsyncTask大展身手的地方。如果创建了大量的AsyncTask，或者长时间在运行AsyncTask，那么很可能就是错用了它。

有一个技术层面的理由更让人信服：在Android 3.2系统版本中，AsyncTask的内部实现有了重大变化。自Android 3.2版本起，AsyncTask不再为每一个AsyncTask实例单独创建线程。相反，它使用一个Executor在单一的后台线程上运行所有AsyncTask后台任务。这意味着每个AsyncTask都需要排队顺序执行。显然，长时间运行的AsyncTask会阻塞其他AsyncTask。

使用一个线程池executor虽然可安全地并发运行多个AsyncTask，但不推荐这么做。如果真的考虑这么做，最好自己处理线程相关的工作，必要时就使用Handler与主线程通信。

26.7　深入学习：解决图片下载问题

本书使用的都是Android官方库中的工具。如有需要，还可以使用各种第三方库。这些库专

用于一些特定场景（比如PhotoGallery中的图片下载），可以节约大量开发时间。

必须承认，PhotoGallery应用的图片下载解决方案远不够完美。如果还想优化性能，实现棘手的缓存功能，很自然就会想到是否别人已有更好的解决方案。答案是肯定的。有好几个高性能图片下载库可供使用。例如，在开发生产应用时，我们就用了Picasso库（square.github.io/ picasso/ ）。

使用Picasso库，一条语句就能实现本章的图片下载功能：

```
private class PhotoHolder extends RecyclerView.ViewHolder {
       ...
        public void bindGalleryItem(GalleryItem galleryItem) {
            Picasso.with(getActivity())
                    .load(galleryItem.getUrl())
                    .placeholder(R.drawable.bill_up_close)
                    .into(mItemImageView);
        }
        ...
}
```

上述代码中，流接口需要使用`with(Context)`指定一个context。`load(String)`用于指定要下载图片的URL。`into(ImageView)`用于指定加载下载结果的`ImageView`对象。当然，还有一些其他配置选项可用，比如指定占位图片（使用`placeholder(int)`和`placeholder(drawable)`）。

在PhotoGallery应用中，只要引入Picasso依赖库，并在`PhotoAdapter.onBindViewHolder(...)`方法中用`bindGalleryItem(...)`方法替换原有代码，就用上了Picasso库的强大下载功能。

Picasso包办了`ThumbnailDownloader`（还有`ThumbnailDownloader.ThumbnailDownload-Listener<T>`回调方法）的所有工作以及`FlickrFetchr`中的图片处理相关工作，所以可以直接删除`ThumbnailDownloader`实现（`FlickrFetchr`中的JSON数据下载还是需要的）。使用Picasso，不仅能简化代码，还能轻松使用它的图片动画、磁盘缓存等高级功能。

可以在项目结构窗口中将Picasso作为库依赖项添加在项目中，就像添加RecyclerView等其他依赖项一样。

当然，Picasso也不是万能的，为追求小而美，它也有功能取舍，比如，它无法支持下载动态图片。如果你有这个需求，可以考虑使用Google的Glide或Facebook的Fresco。它们各有特点，Glide比较小巧，Fresco性能好。

26.8 深入学习：StrictMode

开发应用时，有些东西最好要避免，比如，让应用崩溃的代码漏洞、安全漏洞等。举例来讲，网络条件不好的情况下，在主线程上发送网络请求很可能就会导致设备出现ANR错误。

表现在后台的话，你应该会看到`NetworkOnMainThread`异常以及其他大量日志信息。这实际是StrictMode就错误在警告你。Android引入的StrictMode可以帮助开发者探测代码问题。像在主线程上发起网络请求、编码漏洞以及安全漏洞这样的问题都是它探测的对象。

无需配置，StrictMode就会阻止在主线程上发起网络请求这样的代码问题。它还能探测影响系统性能的代码问题。想启用StrictMode默认防御策略的话，调用`StrictMode.enableDefaults()`

方法就行了（developer.android.com/reference/android/os/StrictMode.html#enableDefaults()）。

一旦调用了`StrictMode.enableDefaults()`方法，如果代码有相关问题，就能在Logcat看到以下提醒：

- 在主线程上发起网络请求
- 在主线程上做了磁盘读写
- Activity未及时销毁（又称为activity泄露）
- SQLite数据库游标未关闭
- 网络通信使用了明文（未使用SSL/TLS加密）

假如应用违反了防御策略，你想定制应对行为，可使用`ThreadPolicy.Builder`和`VmPolicy.Builder`类定制。你可以定制的应对行为有：控制是否抛出异常，弹出对话框或是日志记录违反策略警示信息。

26.9　挑战练习：预加载以及缓存

应用中并非所有任务都能即时完成，对此，大多用户表示理解。不过，即使是这样，开发者们也一直在努力做到最好。

为了让应用反应更快，大多数严肃应用都可以采用以下方式：增加缓存层，预加载图片。

缓存指存储一定数目`Bitmap`对象的地方。这样，即使不再使用这些对象，它们也依然存储在那里。缓存的存储空间有限，因此，在缓存空间用完的情况下，需要某种策略对保存的对象做一定的取舍。许多缓存机制使用一种叫作LRU（least recently used，最近最少使用）的存储策略。基于该种策略，当存储空间用尽时，缓存会清除最近最少使用的对象。

Android支持库中的`LruCache`类实现了LRU缓存策略。作为第一个练习，请使用`LruCache`为`ThumbnailDownloader`增加简单的缓存功能。这样，每次下载完`Bitmap`时，将其存入缓存中。随后，准备下载新图片时，应首先查看缓存，确认是否已经有了。

缓存实现完成后，即可使用它进行预加载。预加载是指在实际使用对象前，就预先将它加载到缓存中。这样，在显示`Bitmap`时，就不会存在下载延迟。

完美的预加载虽然不容易做，但对用户来说，这会带来截然不同的使用体验。作为第二个稍有难度的练习，请在显示`GalleryItem`时，为前十个和后十个`GalleryItem`预加载`Bitmap`。

第 27 章

搜　　索

本章，我们为PhotoGallery应用添加搜索功能。借此学习如何使用SearchView在应用中整合搜索功能。SearchView是个可以嵌入工具栏的操作视图类（action view）。

点按SearchView，用户可以输入查询关键字，提交查询请求搜索Flickr，返回结果将显示在RecyclerView中，如图27-1所示。用户提交过的查询关键字会保存下来。这样，即便应用或设备重启，依然可以找回他们。

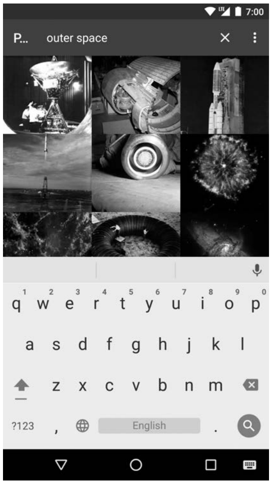


图27-1　搜索界面

27.1　搜索 Flickr 网站

搜索Flickr网站需要调用flickr.photos.search方法。以下为搜索cat文本的GET请求示例：

```
https://api.flickr.com/services/rest/?method=flickr.photos.search
&api_key=xxx&format=json&nojsoncallback=1&text=cat
```

可以看到，搜索方法指定为flickr.photos.search。一个text新参数附加在请求后面，它的内容就是cat这样的搜索字符串。

虽然搜索URL和图片下载请求URL不同，但网站返回的JSON数据格式是一样的。这方便了开发，因为不管是搜索还是下载图片，我们都可以使用同样的JSON数据解析逻辑。

首先，重构FlickrFetchr代码以复用JSON数据解析逻辑。先添加一些URL复用相关的常量。再从fetchItems方法中剪切URI创建代码，复制作为ENDPOINT的值。注意，只应使用加亮部分的代码。ENDPOINT常量不应包括查询方法参数，build语句不应使用toString()方法。如代码清单27-1所示。

代码清单27-1　添加URL常量（FlickrFetchr.java）

```
public class FlickrFetchr {

    private static final String TAG = "FlickrFetchr";

    private static final String API_KEY = "yourApiKeyHere";
    private static final String FETCH_RECENTS_METHOD = "flickr.photos.getRecent";
    private static final String SEARCH_METHOD = "flickr.photos.search";
    private static final Uri ENDPOINT = Uri
            .parse("https://api.flickr.com/services/rest/")
            .buildUpon()
            .appendQueryParameter("api_key", API_KEY)
            .appendQueryParameter("format", "json")
            .appendQueryParameter("nojsoncallback", "1")
            .appendQueryParameter("extras", "url_s")
            .build();
    ...
    public List<GalleryItem> fetchItems() {

        List<GalleryItem> items = new ArrayList<>();

        try {
            String url = Uri.parse("https://api.flickr.com/services/rest/")
                    .buildUpon()
                    .appendQueryParameter("method", "flickr.photos.getRecent")
                    .appendQueryParameter("api_key", API_KEY)
                    .appendQueryParameter("format", "json")
                    .appendQueryParameter("nojsoncallback", "1")
                    .appendQueryParameter("extras", "url_s")
                    .build().toString();
            String jsonString = getUrlString(url);
            ...
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (JSONException je) {
            Log.e(TAG, "Failed to parse JSON", je);
        }

        return items;
    }
    ...
}
```

（刚才的代码调整会导致fetchItems()方法出错。没关系，暂时忽略，稍后会重写这个方法。）

为了通用，重命名fetchItems()方法为downloadGalleryItems(String url)。新方法也不应是公共的了，所以改成私有方法，如代码清单27-2所示。

代码清单27-2 重构Flickr代码（FlickrFetchr.java）

```
public class FlickrFetchr {
    ...
    public List<GalleryItem> fetchItems() {
    private List<GalleryItem> downloadGalleryItems(String url) {

        List<GalleryItem> items = new ArrayList<>();

        try {
            String jsonString = getUrlString(url);
            Log.i(TAG, "Received JSON: " + jsonString);
            JSONObject jsonBody = new JSONObject(jsonString);
            parseItems(items, jsonBody);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (JSONException je) {
            Log.e(TAG, "Failed to parse JSON", je);
        }

        return items;
    }
    ...
}
```


downloadGalleryItems(String)新方法使用URL参数，不用再创建URL了。所以，在其内部添加一个新方法基于用户功能（搜索或下载）和查询值创建URL，如代码清单27-3所示。

代码清单27-3 添加创建URL的辅助方法（FlickrFetchr.java）

```
public class FlickrFetchr {
    ...
    private List<GalleryItem> downloadGalleryItems(String url) {
        ...
    }

    private String buildUrl(String method, String query) {
        Uri.Builder uriBuilder = ENDPOINT.buildUpon()
                .appendQueryParameter("method", method);

        if (method.equals(SEARCH_METHOD)) {
            uriBuilder.appendQueryParameter("text", query);
        }

        return uriBuilder.build().toString();
    }

    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
            throws IOException, JSONException {
        ...
    }
}
```

如同已删除的fetchItems()方法，buildUrl(...)方法会自动拼接必要的参数。不过，参数值是动态确定的。如果判断出是搜索，它就会附加一个text参数值。

现在为下载和搜索添加相应的方法，如代码清单27-4所示。

代码清单27-4　添加方法用于下载和搜索（FlickrFetchr.java）

```
public class FlickrFetchr {
    ...
    public String getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public List<GalleryItem> fetchRecentPhotos() {
        String url = buildUrl(FETCH_RECENTS_METHOD, null);
        return downloadGalleryItems(url);
    }

    public List<GalleryItem> searchPhotos(String query) {
        String url = buildUrl(SEARCH_METHOD, query);
        return downloadGalleryItems(url);
    }

    private List<GalleryItem> downloadGalleryItems(String url) {
        List<GalleryItem> items = new ArrayList<>();
        ...
        return items;
    }
    ...
}
```

FlickrFetchr现在可以处理搜索和图片下载了。fetchRecentPhotos()和searchPhotos(String)方法各自担任从Flickr获取GalleryItem的公共接口。

FlickrFetchr方法中的重构应体现在fragment代码中。在PhotoGalleryFragment.java中，更新FetchItemsTask类代码。

代码清单27-5　硬编码的搜索字符串（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void,List<GalleryItem>> {
        @Override
        protected List<GalleryItem> doInBackground(Void... params) {

            return new FlickrFetchr().fetchItems();
            String query = "robot"; // Just for testing

            if (query == null) {
                return new FlickrFetchr().fetchRecentPhotos();
            } else {
                return new FlickrFetchr().searchPhotos(query);
            }
        }
```

```
            @Override
            protected void onPostExecute(List<GalleryItem> items) {
                mItems = items;
                setupAdapter();
            }
        }
    }
```

如果搜索查询字符串非空（现在肯定非空），FetchItemsTask就会执行Flickr搜索任务。否则，就和以前一样，默认下载最新公共图片。

尽管还没有为用户提供输入查询的用户界面，但我们可以使用硬编码搜索字符串来测试搜索代码。

运行PhotoGallery并查看返回结果。应该可以看到一两张机器人图片，如图27-2所示。

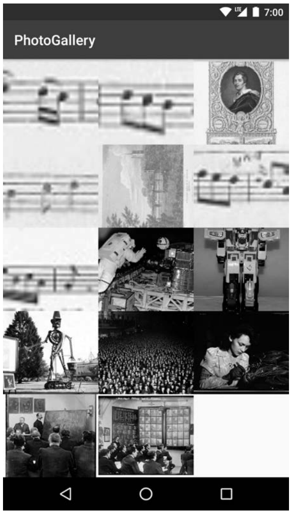


图27-2　硬编码搜索结果

27.2　使用 SearchView

既然FlickrFetchr已支持搜索，现在就来用SearchView创建搜索界面，让用户输入查询关键字并触发搜索。

SearchView是个操作视图。所谓操作视图，就是可以内置在工具栏中的视图。SearchView可以让整个搜索界面完全内置在应用的工具栏中。

首先，确认应用顶部有工具栏（包含应用名称）。如果没有，请参照第13章添加。

接下来，在res/menu/fragment_photo_gallery.xml文件中，为PhotoGalleryFragment创建一个新的菜单XML文件，可以通过这个文件指定工具栏上要显示什么。如代码清单27-6所示。

27

代码清单27-6　添加菜单XML文件（res/menu/fragment_photo_gallery.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/menu_item_search"
          android:title="@string/search"
          app:actionViewClass="android.support.v7.widget.SearchView"
          app:showAsAction="ifRoom" />

    <item android:id="@+id/menu_item_clear"
          android:title="@string/clear_search"
          app:showAsAction="never" />
</menu>
```

新XML文件会出现一些错误，因为暂时还没有为android:title属性定义字符串。忽略这些，稍后会处理。

通过为app:actionViewClass属性指定android.support.v7.widget.SearchView值，代码清单27-6中的第一个定义项告诉工具栏要显示SearchView。（注意，showAsAction和actionViewClass属性都需要app命名空间。不清楚为什么这样用的话，建议再复习一下第13章内容。）

SearchView（android.widget.SearchView）最早是在API 11中（Honeycomb 3.0）引入的。不过，现在它已放入支持库中（android.support.v7.widget.SearchView）。那么到底该用哪个版本呢？相信你已在代码中看到答案：使用支持库版本。是不是有点奇怪？毕竟PhotoGallery应用最低SDK版本已经是19了。

基于与第7章一样的理由，我们推荐使用支持库版本。随着Android新版本的发表，新功能也在不断添加。这些新功能通常会加入支持库中。主题就是这样的一个例子。Lollipop 5.0（API 21）发布后，原生SearchView有许多选项可以定制SearchView界面风格。要想在早期版本Android系统（最低到API 7）上使用这些新特性，就只能选择支持库版SearchView了。

代码清单27-6中的第二个定义项会添加一个Clear Search选项。由于app:showAsAction属性值设置为never，这个选项就只能出现在溢出菜单中。后面，我们会配置它，实现点按该选项就删除已保存的搜索字符串。所以，暂时先忽略它。

现在来解决菜单XML中的未定义字符串错误。打开strings.xml文件，添加缺失的字符串，如代码清单27-7所示。

代码清单27-7　添加搜索字符串（res/values/strings.xml）

```
<resources>
        ...
        <string name="search">Search</string>
        <string name="clear_search">Clear Search</string>

</resources>
```

最后，在PhotoGalleryFragment.java文件中，在onCreate(...)方法中调用setHasOptionsMenu(true)方法让fragment接收菜单回调方法。然后，覆盖onCreateOptionsMenu(...)方法并实例

化菜单XML文件，如代码清单27-8所示。这样，工具栏就能显示定义在菜单XML中的选项了。

代码清单27-8　覆盖onCreateOptionsMenu(...)方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        setHasOptionsMenu(true);
        new FetchItemsTask().execute();
        ...
    }
    ...
    @Override
    public void onDestroy() {
        ...
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        super.onCreateOptionsMenu(menu, menuInflater);
        menuInflater.inflate(R.menu.fragment_photo_gallery, menu);
    }

    private void setupAdapter() {
        ...
    }
    ...
}
```


运行PhotoGallery看看SearchView的界面是什么样的。点击Search按钮，会出现一个供用户输入的文本框，如图27-3所示。



图27-3　搜索界面

SearchView展开后，一个x按钮会出现在右边。点按它会删除用户输入文字。再次点按它，SearchView就会回到只有一个搜索按钮的界面。

现在尝试提交搜索不会有任何结果。不要急，SearchView稍后就会有响应。

响应用户搜索

用户提交查询后，应用立即开始搜索Flickr网站，然后刷新显示搜索结果。查阅开发文档可知，SearchView.OnQueryTextListener接口已提供了接收回调的方式，可以响应查询指令。

更新onCreateOptionsMenu(...)方法，添加一个SearchView.OnQueryTextListener监听方法，如代码清单27-9所示。

代码清单27-9　日志记录SearchView.OnQueryTextListener事件（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        super.onCreateOptionsMenu(menu, menuInflater);
        menuInflater.inflate(R.menu.fragment_photo_gallery, menu);

        MenuItem searchItem = menu.findItem(R.id.menu_item_search);
        final SearchView searchView = (SearchView) searchItem.getActionView();

        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(String s) {
                Log.d(TAG, "QueryTextSubmit: " + s);
                updateItems();
                return true;
            }

            @Override
            public boolean onQueryTextChange(String s) {
                Log.d(TAG, "QueryTextChange: " + s);
                return false;
            }
        });
    }

    private void updateItems() {
        new FetchItemsTask().execute();
    }
    ...
}
```

在onCreateOptionsMenu(...)方法中，我们首先从菜单中取出MenuItem并把它保存在searchItem变量中。然后，使用getActionView()方法从这个变量中取出SearchView对象。

取到SearchView对象，就可以使用setOnQueryTextListener(...)方法设置SearchView.OnQueryTextListener了。另外，你还必须覆盖监听器接口实现里的onQueryTextSubmit

(String)和onQueryTextChange(String)方法。

只要SearchView文本框里的文字有变化，onQueryTextChange (String)回调方法就会执行。在PhotoGallery应用中，这个回调方法除了记日志以外不会干其他任何事。

用户提交搜索查询时，onQueryTextSubmit(String)回调方法就会执行。用户提交的搜索字符串会传给它。搜索请求受理后，该方法会返回true。这个方法也是启动FetchItemsTask搜索结果的地方。（现在FetchItemsTask里仍是一个硬编码的查询，稍后会改用用户提交的查询请求。）

updateItems()方法现在还没多大用。稍后，会有好几个地方要执行FetchItemsTask。updateItems()就是一个调用FetchItemsTask的封装方法。

最后，在onCreate(...)方法中，删除创建和执行FetchItemsTask的那行代码，改用updateItems()封装方法，如代码清单27-10所示。

代码清单27-10　使用updateItems()封装方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        setHasOptionsMenu(true);
        new FetchItemsTask().execute();
        updateItems();
        ...
        Log.i(TAG, "Background thread started");
    }
    ...
}
```

运行应用并提交搜索查询。可以看到，虽然图片重新加载了，搜索结果仍然是基于代码清单27-5中的硬编码搜索字符串。另外，在日志中可以看出SearchView.OnQueryTextListener回调方法已成功执行。

注意：如果在模拟器上使用物理键盘（比如笔记本的键盘）提交查询，搜索会连续执行两次。从用户角度看，就是先看到下载的搜索结果，然后这些图片又全部重新加载一次。这是SearchView的一个bug。这个问题只会出现在模拟器上，可以不用管它。

27.3　使用 shared preferences 实现轻量级数据存储

现在，删除硬编码搜索字符串，我们来实现用户在SearchView中输入并提交的查询指令。

在PhotoGallery应用中，一次只有一个激活的查询。应用应该保存这个查询，即使应用或设备重启也不会丢失。要实现这个目标，可以把查询字符串写入shared preferences。只要用户提交查询，就把它写入shared preferences，覆盖掉之前记录的字符串。实际搜索Flickr时，就从shared preferences中取出查询字符串，把它作为text参数值。

shared preferences本质上就是文件系统中的文件，可使用SharedPreferences类读写它。SharedPreferences实例用起来更像一个键值对仓库（类似于Bundle），但它可以通过持久化存储保存数据。键值对中的键为字符串，而值是原子数据类型。进一步查看shared preferences文件可知，它们实际上是一种简单的XML文件，但SharedPreferences类已屏蔽了读写文件的实现细节。shared preferences文件保存在应用沙盒中，所以，不应用它保存类似密码这样的敏感信息。

要获得SharedPreferences定制实例，可使用Context.getSharedPreferences(String, int)方法。然而，在实际开发中，我们并不关心SharedPreferences实例具体是什么样，只要它能共享于整个应用就可以了。这种情况下，最好使用PreferenceManager.getDefaultSharedPreferences(Context)方法，该方法会返回具有私有权限和默认名称的实例（仅在当前应用内可用）。

如代码清单27-11所示，添加一个名为QueryPreferences的新类，用于读取和写入查询字符串。

代码清单27-11　管理保存的查询字符串（QueryPreferences.java）

```
public class QueryPreferences {
    private static final String PREF_SEARCH_QUERY = "searchQuery";

    public static String getStoredQuery(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
                .getString(PREF_SEARCH_QUERY, null);
    }

    public static void setStoredQuery(Context context, String query) {
        PreferenceManager.getDefaultSharedPreferences(context)
                .edit()
                .putString(PREF_SEARCH_QUERY, query)
                .apply();
    }
}
```

PREF_SEARCH_QUERY用作查询字符串的存储key，读取和写入都要用到它。

getStoredQuery(Context)方法返回shared preferences中保存的查询字符串值。不过，它首先要找到指定context中的默认SharedPreferences。（QueryPreferences类没有自己的Context，所以该方法的调用者必须传入一个。）

取出查询字符串值非常简单，调用SharedPreferences.getString(...)就可以了。如果是其他类型数据，就调用对应的取值方法，比如getInt(...)。SharedPreferences.getString(PREF_SEARCH_QUERY,null)方法的第二个参数指定默认返回值，如果找不到PREF_SEARCH_QUERY对应的值，就返回null值。

setStoredQuery(Context)方法向指定context的默认shared preferences写入查询值。在以上代码中，调用SharedPreferences.edit()方法，可获取一个SharedPreferences.Editor实例。它就是在SharedPreferences中保存查询信息要用到的类。与FragmentTransaction的使用类似，利用SharedPreferences.Editor，可将一组数据操作放入一个事务中。如有一批数据要更

新，在一个事务中批量写入就可以了。

完成所有数据的变更准备后，调用SharedPreferences.Editor的apply()异步方法写入数据。这样，该SharedPreferences文件的其他用户就能看到写入的数据了。apply()方法首先在内存中执行数据变更，然后在后台线程上真正把数据写入文件。

QueryPreferences是PhotoGallery应用的数据存储引擎。既然已经搞定了查询信息的读取和写入方法，现在就在PhotoGalleryFragment中应用它们。

首先是保存用户提交的查询信息，如代码清单27-12所示。

代码清单27-12　存储用户提交的查询信息（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        ...
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(String s) {
                Log.d(TAG, "QueryTextSubmit: " + s);
                QueryPreferences.setStoredQuery(getActivity(), s);
                updateItems();
                return true;
            }

            @Override
            public boolean onQueryTextChange(String s) {
                Log.d(TAG, "QueryTextChange: " + s);
                return false;
            }
        });
    }
    ...
}
```

27

接下来，在用户从溢出菜单选择Clear Search选项时清除存储的查询信息（设置为null），如代码清单27-13所示。

代码清单27-13　清除查询信息（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        ...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_item_clear:
```

```
                QueryPreferences.setStoredQuery(getActivity(), null);
                updateItems();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
    ...
}
```

注意到了没有？和代码清单27-12中的做法一样，更新完查询信息，updateItems()方法会被调用。这很有必要，可以确保RecyclerView中显示最新的搜索结果。

最后，别忘了更新FetchItemsTask，以使用保存的查询字符串（终于可以不用硬编码字符串了）。在FetchItemsTask中添加一个定制构造方法，用于接收查询信息并保存在一个成员变量中备用。更新updateItems()方法，从shared preferences中取出保存的查询信息，用它创建一个FetchItemsTask新实例，如代码清单27-14所示。

代码清单27-14　在FetchItemsTask中使用保存的查询信息（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    private void updateItems() {
        String query = QueryPreferences.getStoredQuery(getActivity());
        new FetchItemsTask(query).execute();
    }
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void,List<GalleryItem>> {
        private String mQuery;

        public FetchItemsTask(String query) {
            mQuery = query;
        }

        @Override
        protected List<GalleryItem> doInBackground(Void... params) {
            String query = "robot"; // Just for testing

            if (querymQuery == null) {
                return new FlickrFetchr().fetchRecentPhotos();
            } else {
                return new FlickrFetchr().searchPhotos(querymQuery);
            }
        }

        @Override
        protected void onPostExecute(List<GalleryItem> items) {
            mItems = items;
            setupAdapter();
        }
    }
}
```

搜索功能现在应该能用了。运行PhotoGallery应用，尝试一些搜索并查看返回结果。

27.4　优化应用

本章任务完成，可以考虑做点应用优化了。如果用户点击搜索按钮展开SearchView时，搜索文本框能显示已保存的查询字符串该多好。用户点击搜索按钮时，SearchView的View.OnClickListener.onClick()方法会被调用。利用这个回调方法设置搜索文本框的值，如代码清单27-15所示。

代码清单27-15　默认显示已保存查询信息（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        ...
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            ...
        });

        searchView.setOnSearchClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String query = QueryPreferences.getStoredQuery(getActivity());
                searchView.setQuery(query, false);
            }
        });
    }
    ...
}
```

运行应用，尝试一些搜索。然后，检验一下刚才优化的成果。应用优化无法一蹴而就，关键是要做个有心人。

27.5　挑战练习：深度优化 PhotoGallery 应用

你可能已经注意到了，提交搜索时，RecyclerView要等好一会才能刷新显示搜索结果。请接受挑战，让搜索过程更流畅一些。用户一提交搜索，就隐藏软键盘，收起SearchView视图（回到只显示搜索按钮的初始状态）。

再来个挑战。用户一提交搜索，就清空RecyclerView，显示一个搜索结果加载状态界面（使用状态指示器）。下载到JSON数据之后，就删除状态指示器。也就是说，一旦开始下载图片，就不应显示加载状态了。

第 28 章 后台服务

目前为止，本书所有的应用都离不开activity，也就是说它们都有一个或多个看得见的用户界面。

如果不给应用提供用户界面，应该怎样做呢？如果不用看、不用操作，只想任务在后台运行，如播放音乐或在RSS feed上检查新博文推送，又该如何做呢？好办，使用服务（service）。

本章，我们为PhotoGallery应用再添一项功能，允许其在后台下载新的搜索结果。一旦有了新结果，用户就能在状态栏看到到通知消息。

28.1 创建 IntentService

首先使用IntentService创建服务。IntentService并不是Android唯一可用的服务，但应该是最常用的。创建一个名为PollService的IntentService子类，它就是用来轮询搜索结果的服务。

代码清单28-1 创建PollService（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    public static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }

    public PollService() {
        super(TAG);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.i(TAG, "Received an intent: " + intent);
    }
}
```

上述代码实现了最基本的IntentService。它能做什么呢？实际上，它和activity有点像。它是一个context（Service是Context的子类），能够响应intent（看onHandleIntent(Intent)方法就

知道）。好的规范要遵守，所以，我们添加一个newIntent (Context)方法。无论谁想启动这个服务，都应使用它。

服务的intent又叫命令（command）。所谓命令，就是要服务做事的一条指令。服务的种类不同，其执行命令的方式也不尽相同。

IntentService服务能顺序执行命令队列里的命令，如图28-1所示。

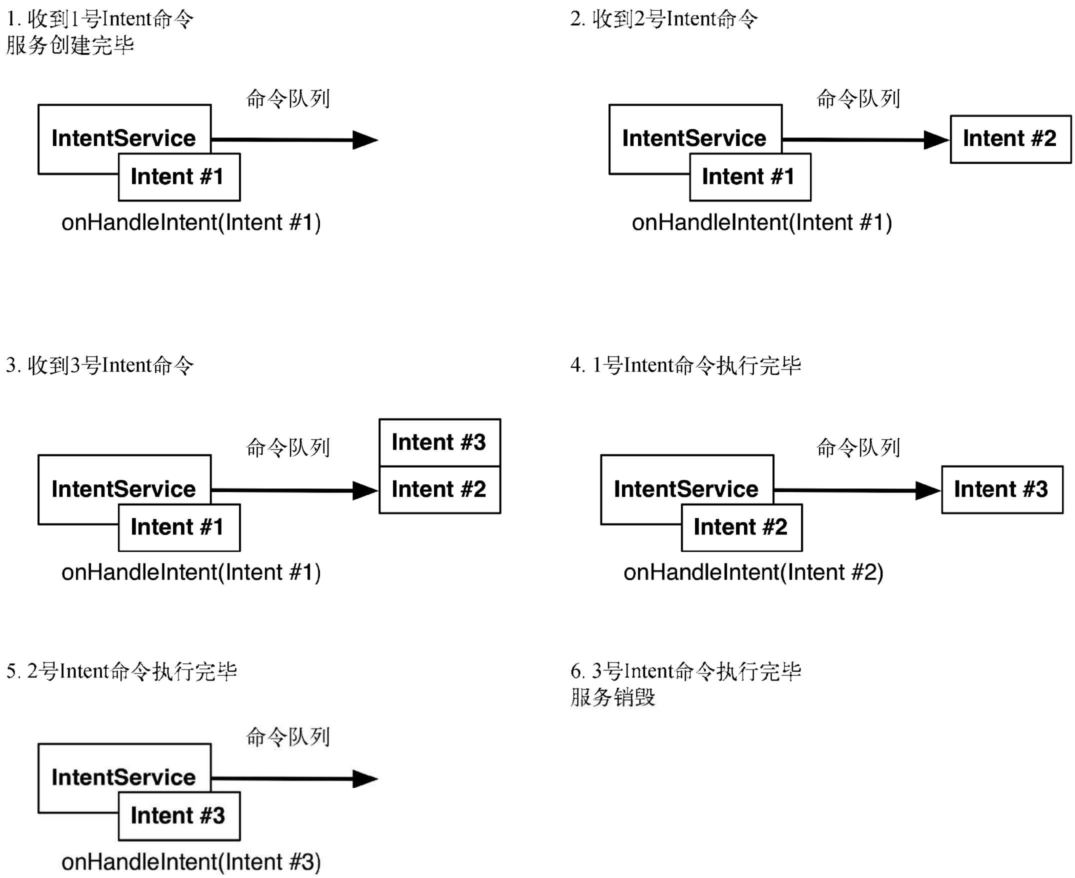


图28-1　IntentService执行命令的方式

收到第一条命令时，IntentService启动，触发一个后台线程，然后将命令放入一个队列。

随后，IntentService按顺序执行每一条命令，并针对每一条命令在后台线程上调用onHandleIntent(Intent)方法。新进命令总是放置在队尾。最后，执行完队列中的全部命令后，服务也随即停止并被销毁。

以上描述仅适用于IntentService。本章稍后会介绍更多服务以及它们执行命令的方式。

既然类似于activity，能够响应intent，就必须在AndroidManifest.xml中声明它。因此，添加一

个PollService元素节点定义，如代码清单28-2所示。

代码清单28-2　在manifest配置文件中添加服务（AndroidManifest.xml）

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery" >

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        ... >
        <activity
            android:name=".PhotoGalleryActivity"
            android:label="@string/app_name" >
            ...
        </activity>
        <service android:name=".PollService" />
    </application>

</manifest>
```

然后，在PhotoGalleryFragment中，添加服务启动代码，如代码清单28-3所示。

代码清单28-3　添加服务启动代码（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        updateItems();

        Intent i = PollService.newIntent(getActivity());
        getActivity().startService(i);

        Handler responseHandler = new Handler();
        mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
        ...
    }
    ...
}
```

运行应用，查看LogCat窗口，可看到以下类似结果。

```
02-23 14:25:32.450    2692-2717/com.bignerdranch.android.photogallery I/PollService:
 Received an intent: Intent { cmp=com.bignerdranch.android.photogallery/.PollService }
```

28.2　服务的作用

查看LogCat日志是不是很乏味？确实！但刚添加的代码着实令人兴奋！为什么？利用它可以完成什么？

再次回到之前的假想之地，在那里，不做开发者，我们是与闪电侠一起工作的鞋店工作人员。鞋店内有两个地方：与客户打交道的前台，以及不与客户接触的后台。

目前为止，所有应用代码都在activity中运行。activity就是Android应用的前台。所有应用代码都专注于提供良好的用户视觉体验。

服务就是Android应用的后台。用户不关心后台发生的一切。即使前台关闭，activity消失好久了，后台服务依然能持续工作。

好了，鞋店的假想可以告一段落了。有服务可以做到而activity做不到的事情吗？有！用户离开当前应用后（打开其他应用或退回主屏幕），服务依然可以在后台运行。

后台网络连接安全

服务会在后台轮询Flickr网站。为保证后台网络连接的安全性，我们需进一步完善代码。Android有关闭后台应用网络连接的功能。如果应用非常耗电，关掉它们就能延长续航时间。

然而，这也意味着在后台连接网络时，需使用ConnectivityManager确认网络连接是否可用。

参照代码清单28-4添加相应的检查代码。

代码清单28-4　检查后台网络的可用性（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        if (!isNetworkAvailableAndConnected()) {
            return;
        }

        Log.i(TAG, "Received an intent: " + intent);
    }

    private boolean isNetworkAvailableAndConnected() {
        ConnectivityManager cm =
                (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE);

        boolean isNetworkAvailable = cm.getActiveNetworkInfo() != null;
        boolean isNetworkConnected = isNetworkAvailable &&
                cm.getActiveNetworkInfo().isConnected();

        return isNetworkConnected;
    }
}
```

检查网络是否可用的逻辑在isNetworkAvailableAndConnected()方法中。使用后台数据设置选项关闭后台数据下载后，所有后台服务也就无法联网了。上述代码中，ConnectivityManager.getActiveNetworkInfo()会返回null值，这等于告诉后台服务，网络不可用，哪怕实际可以。

如果后台服务能够使用网络，它会得到一个代表当前网络连接的android.net.NetworkInfo

28

实例。然后还要调用NetworkInfo.isConnected()方法检查当前网络是否已连接。

如果应用找不到可用网络，或者设备没有连上网，onHandleIntent(...)方法就会直接返回（不会尝试去下载数据了，如果这个方法添加了数据下载代码的话）。这个做法不错：网都连不上，还谈什么数据下载。

不要忘了，要使用getActiveNetworkInfo()方法，还要在manifest配置文件中获取ACCESS_NETWORK_STATE权限，如代码清单28-5所示。

代码清单28-5　获取网络状态权限（AndroidManifest.xml）

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        ... >
        ...
    </application>

</manifest>
```

28.3　查找最新返回结果

后台服务会一直查看最新返回结果，因此它要知道最近一次的获取结果。使用SharedPreferences保存结果值再合适不过了。

更新QueryPreferences以存储最近一次获取图片的ID，如代码清单28-6所示。

代码清单28-6　添加存储图片ID的preference常量（QueryPreferences.java）

```
public class QueryPreferences {
    private static final String PREF_SEARCH_QUERY = "searchQuery";
    private static final String PREF_LAST_RESULT_ID = "lastResultId";

    public static String getStoredQuery(Context context) {
        ...
    }

    public static void setStoredQuery(Context context, String query) {
        ...
    }

    public static String getLastResultId(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
                .getString(PREF_LAST_RESULT_ID, null);
    }
```

```
    public static void setLastResultId(Context context, String lastResultId) {
        PreferenceManager.getDefaultSharedPreferences(context)
                .edit()
                .putString(PREF_LAST_RESULT_ID, lastResultId)
                .apply();
    }
}
```

接下来就是完善服务代码了。以下是需要处理的任务。

(1) 从默认SharedPreferences中获取当前查询结果以及上一次结果ID。

(2) 使用FlickrFetchr类获取最新结果集。

(3) 如果有结果返回，抓取第一条结果。

(4) 确认是否不同于上一次结果ID。

(5) 将第一条结果存入SharedPreferences。

回到PollService.java中，添加实现代码。代码清单28-7中的代码虽长，但相信你已经很熟悉了，这里不再赘述。

代码清单28-7　检查最新返回结果（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        ...
        Log.i(TAG, "Received an intent: " + intent);
        String query = QueryPreferences.getStoredQuery(this);
        String lastResultId = QueryPreferences.getLastResultId(this);
        List<GalleryItem> items;

        if (query == null) {
            items = new FlickrFetchr().fetchRecentPhotos();
        } else {
            items = new FlickrFetchr().searchPhotos(query);
        }

        if (items.size() == 0) {
            return;
        }

        String resultId = items.get(0).getId();
        if (resultId.equals(lastResultId)) {
            Log.i(TAG, "Got an old result: " + resultId);
        } else {
            Log.i(TAG, "Got a new result: " + resultId);
        }

        QueryPreferences.setLastResultId(this, resultId);
    }
    ...
}
```

运行PhotoGallery应用，可看到应用首先获取了最新结果。如选择上一次的搜索查询，提交搜索后，很可能会看到和上次同样的结果。

28.4　使用 AlarmManager 延迟运行服务

在没有activity运行的情况下，为在后台运行服务，得想个办法启动它。比如说，设置一个5分钟间隔的定时器。

一种方式是调用Handler的sendMessageDelayed(...)或postDelayed(...)方法。但如果用户离开当前应用，进程就会停止，Handler消息也会随之消亡，因此这个方案并不可靠。

Handler不行，我们还可以用AlarmManager。AlarmManager是可以发送Intent的系统服务。

如何告诉AlarmManager我想发的intent呢？使用PendingIntent。使用PendingIntent打包一个intent："我想启动PollService服务。"然后，将其发送给系统中的其他部件，如AlarmManager。

在PollService类中，实现一个启停定时器的setServiceAlarm(Context,boolean)方法，该方法是个静态方法。这样，定时器代码和与之相关的代码就可以放在一起了，同时，其他系统部件还可以调用到它。要知道，定时器通常是从前端的fragment或其他控制层代码中启停的。如代码清单28-8所示。

代码清单28-8　添加定时方法（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    // Set interval to 1 minute
    private static final long POLL_INTERVAL_MS = TimeUnit.MINUTES.toMillis(1);

    public static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }

    public static void setServiceAlarm(Context context, boolean isOn) {
        Intent i = PollService.newIntent(context);
        PendingIntent pi = PendingIntent.getService(context, 0, i, 0);

        AlarmManager alarmManager = (AlarmManager)
                context.getSystemService(Context.ALARM_SERVICE);

        if (isOn) {
            alarmManager.setRepeating(AlarmManager.ELAPSED_REALTIME,
                    SystemClock.elapsedRealtime(),POLL_INTERVAL_MS, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }
    }
    ...
}
```

以上代码中，首先是调用PendingIntent.getService(...)方法，创建一个用来启动PollService的PendingIntent。PendingIntent.getService(...)方法打包了一个Context.startService(Intent)方法的调用。它有四个参数：一个用来发送intent的Context，一个区分PendingIntent来源的请求代码，一个待发送的Intent对象以及一组用来决定如何创建PendingIntent的标志符。（稍后会使用其中的一个。）

接下来，需要设置或取消定时器。

设置定时器可调用AlarmManager.setRepeating(...)方法。该方法同样具有四个参数：一个描述定时器时间基准的常量（稍后详述），定时器启动的时间，定时器循环的时间间隔以及一个到时要发送的PendingIntent。

AlarmManager.ELAPSED_REALTIME是基准时间值，这表明我们是以SystemClock.elapsedRealtime()走过的时间来确定何时启动时间的。也就是说，经过一段指定的时间，就启动定时器。假如使用AlarmManager.RTC，启动基准时间就是当前时刻（例如，System.currentTimeMillis()）。也就是说，一旦到了某个固定时刻，就启动定时器。

取消定时器可调用AlarmManager.cancel(PendingIntent)方法。通常，也需同步取消PendingIntent。稍后，你会看到取消PendingIntent也有助于跟踪定时器状态。

添加一些快速测试代码，从PhotoGalleryFragment中启动PollService服务，如代码清单28-9所示。

代码清单28-9 添加定时器启动代码（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        updateItems();

        Intent i = PollService.newIntent(getActivity());
        getActivity().startService(i);
        PollService.setServiceAlarm(getActivity(), true);

        Handler responseHandler = new Handler();
        mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
        ...
    }
    ...
}
```

完成以上代码添加后，运行PhotoGallery应用。然后，立即点击后退键退出应用。

注意观察LogCat日志窗口。可看到PollService服务运行了一次，随后以60秒为间隔再次运行。这正是AlarmManager擅长的事情。即使进程停止了，AlarmManager依然会不断发送intent，反复启动PollService服务。（这种后台服务行为有时非常恼人。要彻底清除它，可能需要卸载应用。）

28.4.1　合理控制服务启动的频度

后台重复性工作的频度需要十分精确吗？后台服务重复性的工作会消耗电池电量和数据流量。而且，启动服务还要唤醒设备，这也是个开销极大的操作。好在这些都可以控制。我们可以采取措施合理配置定时器的启停。比如，确定合理的时间间隔，设置唤醒条件等。

1. 非精准重复

`setRepeating(...)`方法可以设置一个重复定时器，但不是太精准。换句话说，间隔时间可能长一些，也可能短一些，具体Android说了算。一般来说，60秒是目前主流设备支持的最短时间（其他旧设备支持调长一些）。

为啥这样控制呢？因为定时器会给手机电源管理带来麻烦。每次定时器一启动，就会唤醒设备，启动某个应用。类似PhotoGallery，许多应用也需要使用网络，这就更耗电了。

如果只有你一个应用，定时器准不准无所谓。即使定时器每隔15分钟就唤醒设备，那么由于只有一个这样的定时器且不精准，一个小时只需唤醒设备4次。

如果有你一个再加上九个其他应用，每个都有一个15分钟间隔的精准定时器，情况就完全不同了。因为每一个定时都非常精准，都需要唤醒设备，每小时唤醒次数就直接变为恐怖的40次。

上面说过，不精准就意味着Android可以调整定时器的启动时间，这样，定时器就不用非常精准地每隔15分钟启动了。结果，设备唤醒后就可以同时运行这些定时器，每小时唤醒次数又得以降至4次。40次到4次，显然要省太多电了。

当然，有些应用就是需要精确定时器。既然避免不了，那就必须用到`AlarmManager.setWindow(...)`或`AlarmManager.setExact(...)`方法。这两个方法都能设置定时器精准地启动一次。如果需要重复，那就只能自己动动手了。

2. 时间基准选择

另一个要考虑的因素是时间基准值。Android提供了两个选择：`AlarmManager.ELAPSED_REALTIME`和`AlarmManager.RTC`。

`AlarmManager.ELAPSED_REALTIME`使用自最近一次设备重启（包括睡眠时间）开始走过的时间量作为间隔计算基准。既然是基于流逝的相对时间，并且不依赖于时钟时间，`ELAPSED_REALTIME`就成了PhotoGallery应用定时器的最佳选择。（查阅开发者文档可知，Android也首推使用`ELAPSED_REALTIME`。）

`AlarmManager.RTC`使用UTC时间。然而，UTC没有考虑本地时间，而用户认为自己使用的时钟肯定已考虑了本地时间因素。所以，使用RTC作为定时器时间基准同样也要考虑本地时间因素。如果你想设置一个自然时间的定时器，就得自己处理本地时间和RTC基准时间的换算。嫌麻烦，那就干脆使用`ELAPSED_REALTIME`。

无论使用哪个时间基准值，如果设备处于睡眠模式（黑屏状态），即使时间已过，定时器也不会触发。如果想避免这个问题，可使用与已选基准时间对应的`AlarmManager.ELAPSED_REALTIME_WAKEUP`或`AlarmManager.RTC_WAKEUP`常量，让定时器强制唤醒设备。然而，出于节能考虑，应避免使用唤醒选项，除非需要绝对可靠的定时器任务。

28.4.2 PendingIntent

现在来进一步了解前面提及的PendingIntent。PendingIntent是一种token对象。调用PendingIntent.getService(...)方法获取PendingIntent时，你告诉操作系统："请记住，我需要使用startService(Intent)方法发送这个intent。"随后，调用PendingIntent对象的send()方法时，操作系统会按照要求发送原来封装的intent。

PendingIntent真正精妙的地方在于，将PendingIntent token交给其他应用使用时，它是代表当前应用发送token对象的。另外，PendingIntent本身存在于操作系统而不是token里，因此实际上是你在控制着它。如果不顾及别人感受的话，可以在交给别人一个PendingIntent对象后，立即撤销它，让send()方法什么也做不了。

如果使用同一个intent请求PendingIntent两次，得到的PendingIntent仍会是同一个。你可借此测试某个PendingIntent是否已存在，或撤销已发出的PendingIntent。

28.4.3 使用 PendingIntent 管理定时器

一个PendingIntent只能登记一个定时器。这也是isOn值为false时，setServiceAlarm(Context, boolean)方法的工作原理：首先调用AlarmManager.cancel(PendingIntent)方法撤销PendingIntent的定时器，然后撤销PendingIntent。

既然撤销定时器也随即撤消了PendingIntent，可通过检查PendingIntent是否存在来确认定时器激活与否。具体代码实现时，传入PendingIntent.FLAG_NO_CREATE标志给PendingIntent.getService(...)方法即可。该标志表示如果PendingIntent不存在，则返回null，而不是创建它。

28

添加一个名为isServiceAlarmOn(Context)的新方法，并传入PendingIntent.FLAG_NO_CREATE标志，以判断定时器的启停状态，如代码清单28-10所示。

代码清单28-10 添加isServiceAlarmOn()方法（PollService.java）

```
public class PollService extends IntentService {
    ...
    public static void setServiceAlarm(Context context, boolean isOn) {
        ...
    }

    public static boolean isServiceAlarmOn(Context context) {
        Intent i = PollService.newIntent(context);
        PendingIntent pi = PendingIntent
                .getService(context, 0, i, PendingIntent.FLAG_NO_CREATE);
        return pi != null;
    }
    ...
}
```

这里的PendingIntent仅用于设置定时器，因此PendingIntent空值表示定时器还未设置。

28.5 控制定时器

既然可以开关定时器（也能判定其启停状态），接下来就实现在图形界面里控制其开关。首先添加另一菜单项到menu/fragment_photo_gallery.xml，如代码清单28-11所示。

代码清单28-11 添加服务开关（menu/fragment_photo_gallery.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/menu_item_search"
          ... />

    <item android:id="@+id/menu_item_clear"
          ... />

    <item android:id="@+id/menu_item_toggle_polling"
          android:title="@string/start_polling"
          app:showAsAction="ifRoom" />
</menu>
```

然后添加一些字符串资源，一个用于启动polling，一个用于停止polling，如代码清单28-12所示。（后续还需要其他一些字符串资源，如显示在状态栏的通知信息，因此现在也一并完成添加。）

代码清单28-12 添加polling字符串资源（res/values/strings.xml）

```
<resources>
    ...
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>
    <string name="start_polling">Start polling</string>
    <string name="stop_polling">Stop polling</string>
    <string name="new_pictures_title">New PhotoGallery Pictures</string>
    <string name="new_pictures_text">You have new pictures in PhotoGallery.</string>

</resources>
```

删除前面用于启动定时器的测试代码，添加菜单项实现代码，如代码清单28-13所示。

代码清单28-13 菜单项切换实现（PhotoGalleryFragment.java）

```
private static final String TAG = "PhotoGalleryFragment";
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    updateItems();

    PollService.setServiceAlarm(getActivity(), true);

    Handler responseHandler = new Handler();
    ...
```

```
}
...
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_clear:
            QueryPreferences.setStoredQuery(getActivity(), null);
            updateItems();
            return true;
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

现在，应该可以启停定时器了。然而，你可能已经注意到，即使定时器已经启动了，polling选项菜单也总是显示着Start polling。你应该切换选项菜单标题，以便和定时器启停状态匹配。（类似第13章CriminalIntent应用的SHOW SUBTITLE控制。）

在onCreateOptionsMenu(...)方法中，检查定时器的开关状态，然后相应地更新menu_item_toggle_polling的标题文字，反馈正确的信息给用户，如代码清单28-14所示。

代码清单28-14 菜单项切换（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        super.onCreateOptionsMenu(menu, menuInflater);
        menuInflater.inflate(R.menu.fragment_photo_gallery, menu);

        MenuItem searchItem = menu.findItem(R.id.menu_item_search);
        final SearchView searchView = (SearchView) searchItem.getActionView();

        searchView.setOnQueryTextListener(...);

        searchView.setOnSearchClickListener(...);

        MenuItem toggleItem = menu.findItem(R.id.menu_item_toggle_polling);
        if (PollService.isServiceAlarmOn(getActivity())) {
            toggleItem.setTitle(R.string.stop_polling);
        } else {
            toggleItem.setTitle(R.string.start_polling);
        }
    }
    ...
}
```

28

接着，在onOptionsItemSelected(MenuItem)方法中，让PhotoGalleryActivity刷新工具

栏选项菜单，如代码清单28-15所示。

代码清单28-15　让选项菜单失效（PhotoGalleryFragment.java）

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_clear:
            ...
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);
            getActivity().invalidateOptionsMenu();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

现在，选项菜单切换应该可用了。不过，后台服务要真正有用，还有一个地方要完善。

28.6　通知信息

服务已在后台运行并执行指定任务。不过用户对此毫不知情，因此作用有限。

如果服务需要与用户沟通，*通知信息*（notification）会是个不错的选择。通知信息是指显示在通知抽屉上的消息条目，用户可向下滑动它读取。

想要发送通知信息，首先要创建`Notification`对象。类似第12章的`AlertDialog`，`Notification`需使用构造对象来创建。完整的`Notification`至少应包括以下内容。

- 在Lollipop之前的设备上，首次显示通知信息时，在状态栏上显示的ticker text（Lollipop之后，ticker text不再显示在状态栏上，但仍与可访问性服务相关）。
- 在状态栏上显示的图标（在Lollipop之前的设备上，图标在ticker text消失后出现）。
- 代表通知信息自身，是在通知抽屉中显示的一个视图。
- 待触发的`PendingIntent`，用户点击抽屉中的通知信息时触发。

完成`Notification`对象的创建后，可调用`NotificationManager`系统服务的`notify(int, Notification)`方法发送它。

首先是基础代码准备。在PhotoGalleryActivity.java中，添加一个`newIntent(...)`静态方法，如代码清单28-16所示。该静态方法会返回一个可用来启动`PhotoGalleryActivity`的`Intent`实例。（最后，`PollService`会调用这个方法，把返回结果封装在一个`PendingIntent`中，然后设置给通知消息。）

代码清单28-16　添加`newIntent(...)`静态方法（PhotoGalleryActivity.java）

```
public class PhotoGalleryActivity extends SingleFragmentActivity {

    public static Intent newIntent(Context context) {
```

```
        return new Intent(context, PhotoGalleryActivity.class);
    }

    @Override
    protected Fragment createFragment() {
        return PhotoGalleryFragment.newInstance();
    }
}
```

接着，一旦有了新结果，就让PollService通知用户。也就是说，创建一个Notification对象，然后调用NotificationManager.notify(int, Notification)方法，如代码清单28-17所示。

代码清单28-17　添加通知信息（PollService.java）

```
@Override
protected void onHandleIntent(Intent intent) {
    ...
    String resultId = items.get(0).getId();
    if (resultId.equals(lastResultId)) {
        Log.i(TAG, "Got an old result: " + resultId);
    } else {
        Log.i(TAG, "Got a new result: " + resultId);

        Resources resources = getResources();
        Intent i = PhotoGalleryActivity.newIntent(this);
        PendingIntent pi = PendingIntent.getActivity(this, 0, i, 0);

        Notification notification = new NotificationCompat.Builder(this)
                .setTicker(resources.getString(R.string.new_pictures_title))
                .setSmallIcon(android.R.drawable.ic_menu_report_image)
                .setContentTitle(resources.getString(R.string.new_pictures_title))
                .setContentText(resources.getString(R.string.new_pictures_text))
                .setContentIntent(pi)
                .setAutoCancel(true)
                .build();

        NotificationManagerCompat notificationManager =
                NotificationManagerCompat.from(this);
        notificationManager.notify(0, notification);
    }

    QueryPreferences.setLastResultId(this, resultId);
}
```

我们来从上至下解读一下新增代码。首先，调用setTicker(CharSequence)和setSmallIcon(int)方法，配置ticker text和小图标。（注意，以android.R.drawable.ic_menu_report_image包名形式引用的图标资源已内置于Android framework中，所以就没必要再单独放入资源文件夹了。）

然后，配置Notification在下拉抽屉中的外观。虽然可以定制Notification视图的外观和

样式，但使用带有图标、标题以及文字显示区域的标准视图会更容易些。图标的值来自于setSmallIcon(int)方法，而设置标题和显示文字则需分别调用setContentTitle(CharSequence)和setContentText(CharSequence)方法。

接下来，需指定用户点击Notification消息时所触发的动作行为。与AlarmManager类似，这里使用的是PendingIntent。用户在下拉抽屉中点击Notification消息时，传入setContentIntent(PendingIntent)方法的PendingIntent会被触发。调用setAutoCancel(true)方法可调整上述行为。一旦执行了setAutoCancel(true)设置方法，用户点击Notification消息时，该消息就会从消息抽屉中删除。

最后，从当前context（NotificationManagerCompat.from(this)）中取出一个NotificationManagerCompat实例，然后调用NotificationManagerCompat.notify(...)方法贴出消息。传入的整数参数是通知消息的标识符，在整个应用中该值应该是唯一的。如果使用同一ID发送两条消息，则第二条消息会替换掉第一条消息。在实际开发中，这也是进度条或其他动态视觉效果的实现方式。

本章任务到此结束了。运行应用并打开polling服务。不一会儿，应该就会看到状态栏的通知图标。拉开通知抽屉，就会看到后台服务发送的新结果消息。

既然后台服务已能正常工作，那就改用一个更为合理的定时器常量，如代码清单28-18所示。（使用AlarmManager的预定义时间间隔常量，就可以保证在KitKat之前的设备上获得不那么精准的重复定时器行为。）

代码清单28-18　使用更为合理的定时器常量（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    // Set interval to 1 minute
    private static final long POLL_INTERVAL_MS = TimeUnit.MINUTES.toMillis(1);
    private static final long POLL_INTERVAL_MS = TimeUnit.MINUTES.toMillis(15);
    ...
}
```

28.7　挑战练习：可穿戴设备上的通知

创建和管理通知消息时，如果使用的是NotificationCompat和NotificationManagerCompat这两个类，通知信息会自动出现在已和手持设备配对的可穿戴设备上。用户接收到消息后，向左滑动，就能看到在手持设备上打开应用的选项。点击它，即可在手持设备上触发通知的pending intent。

创建一个Android可穿戴设备模拟器并与手持设备配对，测试一下这种通知行为。如果不知道怎么创建可穿戴设备模拟器，请访问developer.android.com求助。

28.8 深入学习：服务之细节

对于大多数服务任务，推荐使用IntentService。但IntentService模式不一定适合所有架构，因此有必要进一步了解并掌握服务，以便自己实现相关服务。做好接受信息轰炸的心理准备。接下来，我们将学习大量有关服务使用的详细内容与复杂细节。

28.8.1 服务的能与不能

与activity一样，服务是一个有生命周期回调方法的应用组件。这些回调方法同样也会在主UI线程上运行，和activity里的一样。

原生服务不能在后台线程上运行。这也是我们推荐使用IntentService的最主要原因。大多数重要服务都需要在后台线程上运行，而IntentService已提供了一套标准支持方案。

下面来看看服务有哪些生命周期回调方法。

28.8.2 服务的生命周期

如果是startService(Intent)方法启动的服务，其生命周期很简单，并有三种生命周期回调方法。

- onCreate(...)方法：服务创建时调用。
- onStartCommand(Intent,int,int)方法：每次组件通过startService(Intent)方法启动服务时调用一次。它有两个整数参数，一个是标识符集，一个是启动ID。标识符集用来表示当前intent发送究竟是一次重新发送，还是一次从没成功过的发送。每次调用onStartCommand(Intent,int,int)方法，启动ID都会不同。因此，启动ID也可用于区分不同的命令。
- onDestroy()方法：服务不再需要时调用。

服务停止时会调用onDestroy()方法。服务停止的方式取决于服务的类型。服务的类型由onStartCommand(...)方法的返回值确定，可能的服务类型有Service.START_NOT_STICKY、START_REDELIVER_INTENT和START_STICKY。

28

28.8.3 non-sticky 服务

IntentService是一种non-sticky服务。non-sticky服务在服务自己认为已完成任务时停止。为获得non-sticky服务，应返回START_NOT_STICKY或START_REDELIVER_INTENT。

通过调用stopSelf()或stopSelf(int)方法，我们告诉Android任务已完成。stopSelf()是个无条件方法。不管onStartCommand(...)方法调用多少次，该方法总是会成功停止服务。

stopSelf(int)是个有条件的方法。该方法需要来自于onStartCommand(...)方法的启动ID。只有在接收到最新启动ID后，该方法才会停止服务。（这也是IntentService的后台工作原理。）

返回START_NOT_STICKY和START_REDELIVER_INTENT有什么不同呢？区别就在于，如果系统需要在服务完成任务之前关闭它，则服务的具体表现会有所不同。START_NOT_STICKY型服务说没就没了；而START_REDELIVER_INTENT型服务则会在资源不再吃紧时，尝试再次启动服务。

选择START_NOT_STICKY还是START_REDELIVER_INTENT，这要看服务对应用有多重要了。如果不重要，就选择START_NOT_STICKY。在PhotoGallery应用中，服务根据定时器的设定重复运行。即使发生问题，也不会有严重后果，因此应选择START_NOT_STICKY，同时，它也是IntentService的默认行为。如有需要，我们也可调用IntentService.setIntentRedelivery(true)方法，改用START_REDELIVER_INTENT。

28.8.4　sticky 服务

sticky服务会持续运行，直到外部组件调用Context.stopService(Intent)方法让它停止。为获得sticky服务，应返回START_STICKY。

sticky服务启动后会持续运行，除非某个组件调用Context.stopService(Intent)方法停止它。如因某种原因需终止服务，可传入一个null intent给onStartCommand(...)方法重启服务。

sticky服务适用于长时间运行的服务，如音乐播放器这种启动后一直运行，直到用户主动停止的服务。即使是这样，也应考虑一种使用non-sticky服务的替代架构方案。sticky服务的管理很不方便，因为很难判断服务是否已启动。

28.8.5　绑定服务

除以上各类服务外，也可使用bindService(Intent,ServiceConnection, int)方法绑定一个服务，以此获得直接调用绑定服务方法的能力。ServiceConnection是代表服务绑定的一个对象，负责接收全部绑定回调方法。

在fragment中，绑定代码可能是这样的：

```
private ServiceConnection mServiceConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className,
            IBinder service) {
        // Used to communicate with the service
        MyBinder binder = (MyBinder)service;
    }

    public void onServiceDisconnected(ComponentName className) {
    }
};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent i = new Intent(getActivity(), MyService.class);
    getActivity().bindService(i, mServiceConnection, 0);
}
```

```
@Override
public void onDestroy() {
    super.onDestroy();
    getActivity().unbindService(mServiceConnection);
}
```

对服务来说，绑定引入了另外两个生命周期回调方法。

- ❑ onBind(Intent)方法：每次绑定服务时调用，返回来自ServiceConnection.onServiceConnected(ComponentName,IBinder)方法的IBinder对象。
- ❑ onUnbind(Intent)方法：服务绑定终止时调用。

1. 本地服务绑定

MyBinder是怎样一种对象呢？如果服务是个本地服务，MyBinder很可能就是本地进程中的一个简单Java对象。通常，MyBinder用于提供一个句柄，以便直接调用服务方法：

```
private class MyBinder extends IBinder {
    public MyService getService() {
        return MyService.this;
    }
}

@Override
public void onBind(Intent intent) {
    return new MyBinder();
}
```

这种模式看上去让人激动。这是Android系统中唯一一处支持组件间直接对话的地方。不过，我们并不推荐此种模式。服务是种高效的单例，与仅使用一个单例相比，使用此种模式显现不出优势。

28

2. 远程服务绑定

绑定更适用于远程服务，因为它们赋予了其他进程中应用调用服务方法的能力。创建远程绑定服务属于高级主题，不在本书讨论范畴。请查阅Android文档中的AIDL指导或Messenger类细节。

28.9 深入学习：JobScheduler 和 JobService

本章，我们已知道如何让AlarmManager、IntentService和PendingIntent相互配合，创建周期性的后台服务。实现一个完全可用的后台服务还有几件事要做：

- ❑ 计划一个周期性任务；
- ❑ 检查周期性任务的运行状态；
- ❑ 检查网络是否可用。

搭配使用一些API，我们也能手工创建一个能够启停、基本可用的后台服务。虽然可行，但工作量很大。

在Lollipop（API 21）系统中，为更好地实现后台服务，Android引入了一个叫JobScheduler的全新API。除了能用来处理各类任务外，JobScheduler还能做到更多：发现没有网络时，它能禁止服务启动；如果请求失败或网络连接受限，它能提供稍后重试机制；它能控制只在设备充电的时候，才允许联网检查是否有新图片。虽然上述某些场景使用AlarmManager和IntentService也可能实现，试试就知道了，那绝对不容易。

除了实现常规后台服务之外，JobScheduler还支持按场景、按条件运行后台服务。下面就来看看它的工作原理。首先，创建一个处理任务的服务（使用JobService子类）。JobService有两个可覆盖方法：onStartJob(JobParameters)和onStopJob(JobParameters)。（注意，以下代码仅供讨论之用，不用找地方输入。）

```
public class PollService extends JobService {
    @Override
     public boolean onStartJob(JobParameters params) {
         return false;
     }

     @Override
     public boolean onStopJob(JobParameters params) {
         return false;
     }
}
```

Android准备好执行任务时，服务就会启动，此时会在主线程上收到onStartJob(...)方法调用。该方法返回false结果表示："交代的任务我已全力去做，现在做完了。"返回true结果则表示："任务收到，正在做，但是还没有做完。"

与IntentService不同，JobService需要你单开新线程，这点比较麻烦。可使用AsyncTask按如下方式创建新线程：

```
private PollTask mCurrentTask;

@Override
public boolean onStartJob(JobParameters params) {
    mCurrentTask = new PollTask();
    mCurrentTask.execute(params);
    return true;
}

private class PollTask extends AsyncTask<JobParameters,Void,Void> {
    @Override
    protected Void doInBackground(JobParameters... params) {
        JobParameters jobParams = params[0];

        // Poll Flickr for new images

        jobFinished(jobParams, false);
        return null;
    }
}
```

任务完成后，就可以调用jobFinished(JobParameters, boolean)方法通知结果。不过，如果该方法的第二个参数传入true的话，就等于说："事情这次做不完了，请计划在下次某个时间继续吧。"

任务运行时也可能会收到onStopJob(JobParameters)调用，表明当前任务需中断。例如，用户通常需要服务在有WiFi连接时才运行。如果任务还在处理，手机就离开了WiFi覆盖区，onStopJob(...)方法就会被调用，也就是说，一切任务应立即停止。

```
@Override
public boolean onStopJob(JobParameters params) {
    if (mCurrentTask != null) {
        mCurrentTask.cancel(true);
    }
    return true;
}
```

调用onStopJob(...)方法就是表明，服务马上就要停掉了。不要抱有幻想，请立即停止手头上的一切事情。这里，返回true表示："任务应该计划在下次继续。"返回false表示："不管怎样，事情到此结束吧，不要计划下次了。"

在manifest配置文件中登记服务时，必须导出它并为它添加权限：

```
<service
    android:name=".PollService"
    android:permission="android.permission.BIND_JOB_SERVICE"
    android:exported="true"/>
```

所谓导出服务，就是把服务暴露出来，但添加的权限控制只有JobScheduler才能运行它。 28

一旦创建了JobService，启动它就非常迅速了。你可以使用JobScheduler检查是否已计划好了任务。

```
final int JOB_ID = 1;

JobScheduler scheduler = (JobScheduler)
    context.getSystemService(Context.JOB_SCHEDULER_SERVICE);

boolean hasBeenScheduled = false;
for (JobInfo jobInfo : scheduler.getAllPendingJobs()) {
    if (jobInfo.getId() == JOB_ID) {
        hasBeenScheduled = true;
    }
}
```

如果还没有，可以创建一个新的JobInfo说明你期望的任务运行时间。嗯，PollServie应该在什么时候运行呢？这样如何：

```
final int JOB_ID = 1;

JobScheduler scheduler = (JobScheduler)
    context.getSystemService(Context.JOB_SCHEDULER_SERVICE);

JobInfo jobInfo = new JobInfo.Builder(
```

```
            JOB_ID, new ComponentName(context, PollService.class))
            .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
            .setPeriodic(1000 * 60 * 15)
            .setPersisted(true)
            .build();
    scheduler.schedule(jobInfo);
```

上述代码计划任务每15分钟运行一次，但前提条件是有WiFi或有不限流量网络可用。调用`setPersisted(true)`方法可保证服务在设备重启后也能按计划运行。还可采用其他方式配置`JobInfo`，具体做法请参阅Android开发者文档。

JobScheduler 和未来的后台服务

本章介绍不使用`JobScheduler`的前提下如何实现后台服务。没办法，只有Lollipop及其之后的系统才支持JobScheduler，而且它也没有支持库版实现。本章基于AlarmManager实现的定时服务是唯一可行的标准库解决方案，可以支持各个Android系统版本。

然而，有个重要的讯息要告诉你，AlarmManager处理后台服务的日子应该不长了。近年，Android平台开发人员有好几个高优先级的目标要实现，其中有个目标是尽量改善电池使用效率，提高设备的续航时间。因此，对于那些使用WiFi、GPS而特别耗电的应用，他们一直在设法能绝对控制各种场景下的任务调度。

这也是这些年来AlarmManager一直不停在变的原因：Android知道开发人员都在用AlarmManager计划后台服务。为让应用运行更加高效，这些个API必须不断地优化。

然而，实际上，就处理后台定时器服务这些任务来说，AlarmManager并不是个合格的选手。用户在做什么呢，是在使用GPS定位，还是在更新应用widget皮肤，它不知道，自然也就无法告诉系统。既然Android系统掌握不了应用使用情况，那它只能毫无差别地处理各个定时器任务，如何合理调度高效节能自然也就无从谈起。

在此压力下，相信一旦JobScheduler普及开来，Android就会立即抛弃AlarmManager。所以，我们建议，只要你觉得可行，应尽早使用JobScheduler。

如果你不打算在将来做痛苦的迁移，现在就想用起来，可以试试第三方兼容库。目前，Evernote的android-job是最好的选择。可以在github.com/evernote/android-job找到它。

28.10　挑战练习：在 Lollipop 设备上使用 `JobService`

请创建另一个`PollService`实现版本。新的`PollService`应该继承`JobService`并使用`JobScheduler`。在`PollService`启动代码中，检查系统版本是否为Lollipop：如果是，就使用`JobScheduler`计划运行`JobService`；如果不是，依然使用`AlarmManager`实现。

28.11　深入学习：sync adapter

你还可以使用sync adapter创建常规的polling网络服务。和前面看到过的adapter不一样，sync

adapter主要用于从某个数据源同步数据（上传、下载或既上传又下载）。不像`JobScheduler`，sync adapter早就有了，所以不用担心新旧系统版本问题。

和`JobScheduler`一样，sync adapter可代替PhotoGallery应用中的`AlarmManger`。不同应用中的同步功能都是默认一起执行的。而且，即使设备重启，也不用重置同步定时器，因为sync adapter会自动处理。

sync adapter还能和操作系统完美整合。你可以设置一个可同步账户，对外暴露应用。然后，用户通过Settings → Accounts菜单来管理应用的同步。当然，这个菜单还可以用来管理其他使用sync adapter的应用账户，如Google自己的一些应用套件，如图28-2所示。

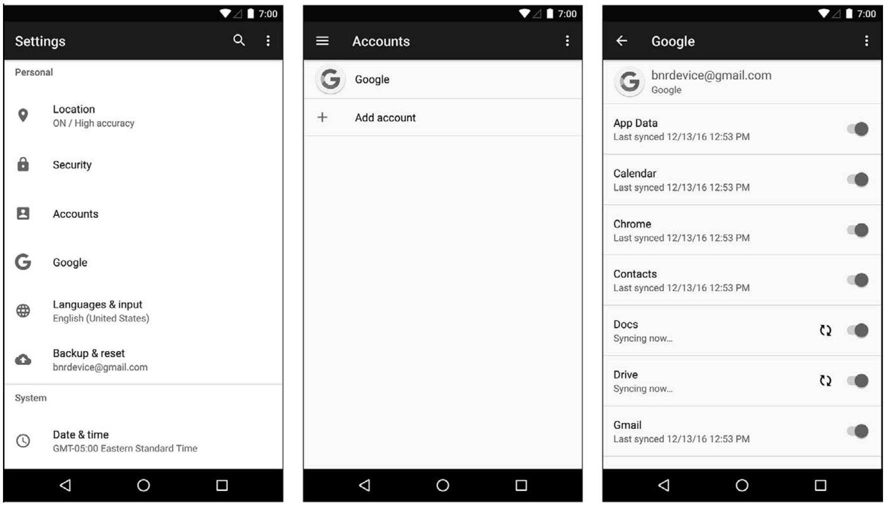


图28-2　账户设置

虽然使用sync adapter既能让周期性的网络任务变得容易可靠，又能让开发者免于编写定时器管理和pending intent的相关代码，但仍然免不了写另外一些代码。首先，需要与网络请求相关的代码（如`FlickFetchr`）。其次，要有一个content provider实现来封装数据、账户和授权类，用以代表远程服务器的某个账户（即使远程服务器不需要授权），以及一个sync adapter和sync service的实现。另外，还要懂得运用绑定服务。

所以，如果应用已经使用content provider作为数据层，并且需要账号授权，那使用sync adapter是最理想的。此外，相较于`JobScheduler`，sync adapter还有同操作系统提供的用户界面自然整合的优势。考虑到这些因素，即使要写一大堆代码，某些场景下，仍然值得使用sync adapter。

在线开发者文档提供了sync adapter使用教程：developer. android.com/training/sync-adapters/index.html。

第 29 章 broadcast intent

本章，我们继续从两个方面完善PhotoGallery应用。首先，让应用轮询新结果并在有所发现时及时通知用户，即使用户重启设备后还没有打开过应用。其次，保证用户在使用应用时不出现新结果通知。（用户开了应用，已看到刷新显示的新的搜索图片，同时还收到了新结果通知，你说是不是既多余又烦人？）

借此升级，我们将学习如何监听系统发送的broadcast intent，以及如何使用broadcast receiver处理它们。此外，我们会在应用运行时动态发送与接收broadcast intent。最后，还会使用有序broadcast判断应用是否在前台运行。

29.1 普通 intent 和 broadcast intent

Android设备中，各种事件时有发生。WiFi时有时无，软件装卸，电话接打，短信收发，等等。

许多系统组件需要掌握这些动态。为满足这样的需求，Android提供了broadcast intent组件。broadcast intent的工作原理类似之前学过的intent，唯一不同的是broadcast intent可同时被多个叫作broadcast receiver的组件接收（如图29-1所示）。

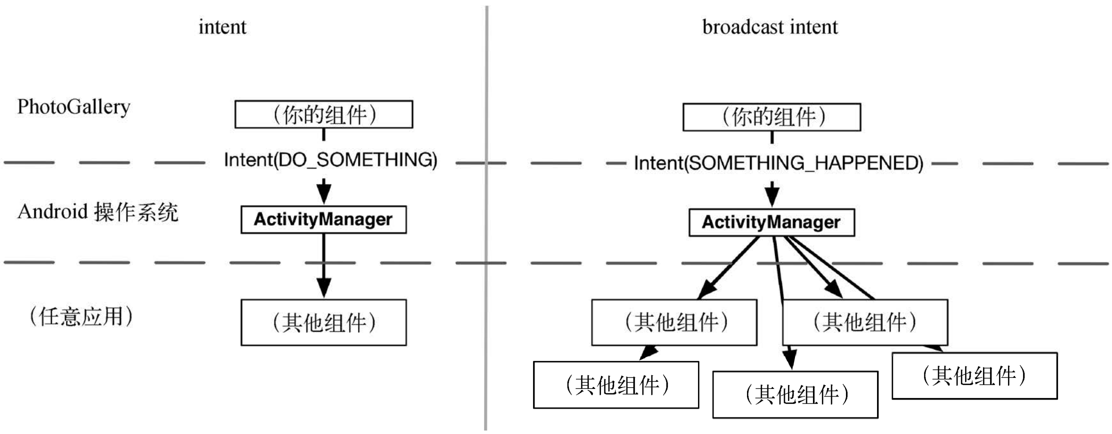


图29-1　普通intent与broadcast intent

作为公共API的一部分，无论什么时候，activity和服务应该都可以响应隐式intent。如果是用作私有API，用用显式intent差不多也够了。这样看来，还需要broadcast intent的话，理由只有一个：它可以发送给多个接收者。虽然broadcast receiver也能响应显式intent，但几乎没人这么用，因为显式intent只允许有一个接收者。

29.2 接收系统 broadcast：重启后唤醒

PhotoGallery应用的后台定时器虽然能用，但还不够完美。如果用户重启了设备，定时器就会失效。

设备重启后，那些持续运行的应用通常也需要重启。监听带有`BOOT_COMPLETED`操作的broadcast intent，可知道设备是否已完成启动。无论何时，只要打开设备，系统就会发送一个`BOOT_COMPLETED` broadcast intent。要想监听它，可以创建并登记一个standalone broadcast receiver。

29.2.1 创建并登记 standalone receiver

standalone receiver是一个在manifest配置文件中声明的broadcast receiver。即便应用进程已消亡，standalone receiver也可以被激活。（稍后还会学习到可以同fragment或activity的生命周期绑定的dynamic receiver。）

与服务和activity一样，broadcast receiver必须在系统中登记后才能用。如果不登记，系统就不知道该向哪里发送intent。自然，broadcast receiver的`onReceive(...)`方法也就不能按预期被调用了。

要登记broadcast receiver，首先要创建它。创建一个`StartupReceiver`新类，继承`android.content.BroadcastReceiver`类，如代码清单29-1所示。

29

代码清单29-1 第一个broadcast receiver（StartupReceiver.java）

```
public class StartupReceiver extends BroadcastReceiver{
    private static final String TAG = "StartupReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "Received broadcast intent: " + intent.getAction());
    }
}
```

与服务和activity一样，broadcast receiver是接收intent的组件。当有intent发送给`StartupReceiver`时，它的`onReceive(...)`方法会被调用。

打开AndroidManifest.xml配置文件，登记上`StartupReceiver`，如代码清单29-2所示。

代码清单29-2 在manifest文件中添加receiver（AndroidManifest.xml）

```
<manifest ...>

    <uses-permission android:name="android.permission.INTERNET"/>
```

```
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

    <application
        ...>
        <activity
            android:name=".PhotoGalleryActivity"
            android:label="@string/app_name">
            ...
        </activity>
        <service android:name=".PollService"/>

        <receiver android:name=".StartupReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

登记响应隐式intent的standalone receiver和登记服务或activity差不多。我们使用receiver标签并在其中包含相应的intent-filter。StartupReceiver会监听BOOT_COMPLETED操作，而该操作也需要配置使用权限。因此，还需要添加一个相应的uses-permission标签。

在配置文件中完成声明后，即使应用还没运行，只要有匹配的broadcast intent发来，broadcast receiver就会醒来接收。一收到intent，broadcast receiver的onReceive(Context, Intent)方法即开始运行，随后会被销毁（如图29-2所示）。

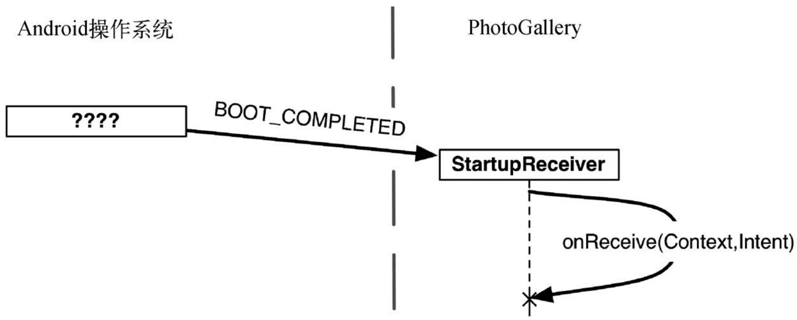


图29-2　接收BOOT_COMPLETED

设备重启后，StartupReceiver的onReceive(...)方法会被调用吗？现在就来验证。首先，运行更新版PhotoGallery应用。

然后，关闭设备。如果是物理设备，按电源键关机。如果是模拟器，关机的最简方法是直接退出模拟器应用。

打开设备。如果是物理设备，按电源键开机。如果是模拟器，要么重新运行应用，要么使用AVD Manager启动应用，但要保证使用的是刚关掉的那个模拟器。

现在，选择Tools → Android → Android Device Monitor菜单项打开Android Device Monitor。

点击Android Device Monitor的Devices选项卡中的设备。（如果看不到设备列表，请尝试插拔USB设备或重启模拟器。）

在Android Device Monitor窗口中，以Received broadcast intent关键字搜索LogCat输出（如图29-3所示）。

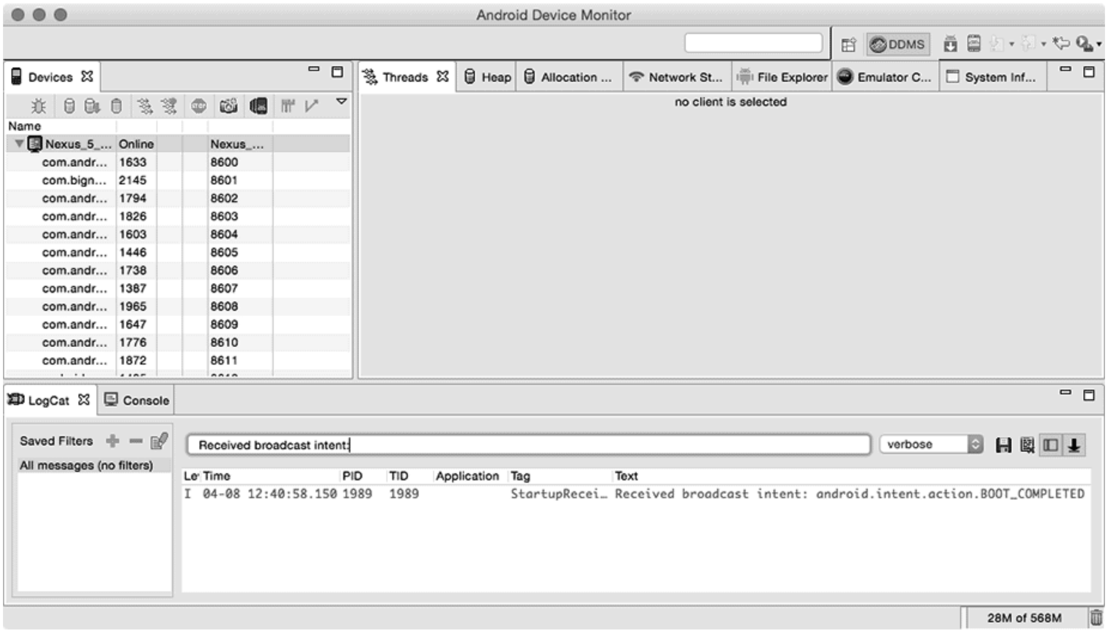


图29-3　搜索LogCat输出

在LogCat中，可以看到表明receiver运行的日志。但如果在设备标签页查看设备，则可能看不到任何PhotoGallery进程。这是因为进程一运行完broadcast receiver，就随即消亡了。

（使用Logcat输出测试receiver的运行不一定总会成功，使用模拟器更有可能。按照上述步骤操作，如果看不到log日志，建议多试几次。还是不行的话，那就暂时放弃，等学习到优化通知消息时，就会有更可靠的办法来验证receiver的运行了。）

29.2.2　使用 receiver

broadcast receiver的生命非常短暂，因而难以有所作为。例如，我们无法使用任何异步API或登记任何监听器，因为一旦`onReceive(Context, Intent)`方法运行完，receiver就不存在了。`onReceive(Context, Intent)`方法同样运行在主线程上，因此不能在该方法内做一些费时费力的事情，如网络连接或数据的永久存储等。

然而，这并不代表receiver一无用处。一些便利型任务就非常适合它，比如启动activity或服

务（不需要等返回结果），以及系统重启后重置定时运行的定时器。

receiver需要知道定时器的启停状态。为存储它的状态，在QueryPreferences类中添加一个preference常量和两个便利方法，如代码清单29-3所示。

代码清单29-3　添加定时器状态preference（QueryPreferences.java）

```
public class QueryPreferences {

    private static final String PREF_SEARCH_QUERY = "searchQuery";
    private static final String PREF_LAST_RESULT_ID = "lastResultId";
    private static final String PREF_IS_ALARM_ON = "isAlarmOn";
    ...
    public static void setLastResultId(Context context, String lastResultId) {
        ...
    }

    public static boolean isAlarmOn(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
                .getBoolean(PREF_IS_ALARM_ON, false);
    }

    public static void setAlarmOn(Context context, boolean isOn) {
        PreferenceManager.getDefaultSharedPreferences(context)
                .edit()
                .putBoolean(PREF_IS_ALARM_ON, isOn)
                .apply();
    }
}
```

接下来，更新PollService.setServiceAlarm(...)方法，在设置定时器后存下它的状态，如代码清单29-4所示。

代码清单29-4　存储定时器状态（PollService.java）

```
public class PollService extends IntentService {
    ...
    public static void setServiceAlarm(Context context, boolean isOn) {
        ...
        if (isOn) {
            alarmManager.setRepeating(AlarmManager.ELAPSED_REALTIME,
                    SystemClock.elapsedRealtime(), POLL_INTERVAL_MS, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }

        QueryPreferences.setAlarmOn(context, isOn);
    }
    ...
}
```

这样，设备重启后，StartupReceiver就能用它打开定时器，如代码清单29-5所示。

代码清单29-5 设备重启后启动定时器（StartupReceiver.java）

```
public class StartupReceiver extends BroadcastReceiver{
    private static final String TAG = "StartupReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "Received broadcast intent: " + intent.getAction());

        boolean isOn = QueryPreferences.isAlarmOn(context);
        PollService.setServiceAlarm(context, isOn);
    }
}
```

再次运行PhotoGallery应用（为方便测试，可设置一个较短的时间间隔，如60秒）。点击工具栏的START POLLING打开服务。重启设备。这次，后台polling服务应该会重启了。

29.3 过滤前台通知消息

解决了设备重启后的服务唤醒问题，再来看PhotoGallery应用的另一缺陷。通知消息虽然很有用，但应用开着的时候不应该收到通知消息。

同样，我们使用broadcast intent来解决这个问题，但用法和以前完全不同。

首先，我们发送（或接收）定制版broadcast intent（最后会锁定它，只允许PhotoGallery应用部件接收它）。其次，不再使用manifest文件，改用代码为broadcast intent动态登记receiver。最后，发送一个有序broadcast在一组receiver中传递数据，借此保证最后才运行某个receiver。（不太理解这段话？别担心，跟着做，很快就会明白了。）

29

29.3.1 发送 broadcast intent

首先处理最容易的部分：发送自己定制的broadcast intent。具体来讲，就是发送broadcast通知目标部件有新的搜索结果消息了。要发送broadcast intent，只需创建一个intent，并传入`sendBroadcast(Intent)`方法即可。这里，需要通过`sendBroadcast(Intent)`方法广播自定义的操作（action），因此还需要定义一个操作常量。

在`PollService`类中，输入代码清单29-6所示代码。

代码清单29-6 发送broadcast intent（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final long POll_iNTERVAl_MS = TimeUnit.MINUTES.toMillis(15);

    public static final String ACTION_SHOW_NOTIFICATION =
            "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
```

```
        ...
        String resultId = items.get(0).getId();
        if (resultId.equals(lastResultId)) {
            Log.i(TAG, "Got an old result: " + resultId);
        } else {
            ...
            NotificationManagerCompat notificationManager =
                    NotificationManagerCompat.from(this);
            notificationManager.notify(0, notification);

            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
        }

        QueryPreferences.setLastResultId(this, resultId);
    }
    ...
}
```

现在，只要有新结果，应用就会对外广播。

29.3.2　创建并登记动态 receiver

完成intent的发送后，接下来的任务是使用receiver接收ACTION_SHOW_NOTIFICATION broadcast intent。

可以编写一个类似于StartupReceiver的standalone broadcast receiver来接收intent，并在manifest文件中登记；但这里行不通。你需要在PhotoGalleryFragment还在活动的时候接收发过来的intent。在配置文件中声明的standalone receiver总在接收intent，而且它还要知道PhotoGalleryFragment的状态（这是个难点）。

使用动态broadcast receiver能解决问题。动态broadcast receiver是在代码中而不是在配置文件中完成登记声明的。要在代码中登记，可调用registerReceiver(BroadcastReceiver, IntentFilter)方法；取消登记时，则调用unregisterReceiver(BroadcastReceiver)方法。receiver自身通常被定义为一个内部类实例，如同一个按钮点击监听器。然而，在registerReceiver(...)和unregisterReceiver(BroadcastReceiver)方法中，你要的是同一个实例，因此需要将receiver赋值给一个实例变量。

新建一个VisibleFragment抽象类，继承Fragment类，如代码清单29-7所示。该类是一个隐藏前台通知的通用型fragment。（在第30章，你还会编写一个类似的fragment。）

代码清单29-7　VisibleFragment自己的receiver（VisibleFragment.java）

```
public abstract class VisibleFragment extends Fragment {
    private static final String TAG = "VisibleFragment";

    @Override
    public void onStart() {
        super.onStart();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter);
```

```
    }

    @Override
    public void onStop() {
        super.onStop();
        getActivity().unregisterReceiver(mOnShowNotification);
    }


    private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getActivity(),
                    "Got a broadcast:" + intent.getAction(),
                    Toast.LENGTH_LONG)
                  .show();
        }
    };
}
```

注意，要传入一个IntentFilter，必须先以代码的方式创建它。这里创建的IntentFilter同以下XML文件定义的filter是一样的：

```
<intent-filter>
    <action android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION" />
</intent-filter>
```

任何使用XML定义的IntentFilter均能以代码的方式定义。要在代码中配置IntentFilter，可以直接调用addCategory(String)、addAction(String)和addDataPath(String)等方法。

使用动态登记的broadcast receiver时，要记得事后清理。通常，如果在生命周期启动方法中登记了一个receiver，就应在相应的停止方法中调用Context.unregisterReceiver(BroadcastReceiver)方法。这里，我们在onStart()方法里登记，在onStop()方法里撤销登记。同样，如果在onCreate(...)方法里登记，就应在onDestroy()里撤销登记。

29

（顺便要说的是，我们应注意在保留fragment中的onCreate(...)和onDestroy()方法的使用。设备旋转时，onCreate(...)和onDestroy()方法中的getActivity()方法会返回不同的值。因此，如果想在Fragment.onCreate(Bundle)和Fragment.onDestroy()方法中实现登记或撤销登记，应改用getActivity().getApplicationContext()方法。）

接下来，修改PhotoGalleryFragment类，转而继承新的VisibleFragment，如代码清单29-8所示。

代码清单29-8 设置fragment为可见（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends ~~Fragment~~ VisibleFragment {
    ...
}
```

运行PhotoGallery应用。多次开关后台结果检查服务，可看到toast提示消息，如图29-4所示。



图29-4　验证broadcast的存在

29.3.3　使用私有权限限制 broadcast

使用动态broadcast receiver存在一个问题，即系统中的任何应用均可监听并触发你的receiver。通常来讲，都不是你想要的。

不要担心，有多种方式可以阻止未授权应用闯入你的私人领域。一种办法是在manifest配置文件里给receiver标签添加一个`android:exported="false"`属性，声明它仅限应用内部使用。这样，系统中的其他应用就再也无法接触到该receiver了。

另外，也可创建自己的使用权限。这需要在AndroidManifest.xml中添加一个`permission`标签。这是PhotoGallery应用要用到的办法。

在AndroidManifest.xml配置文件中，声明并获取自己的使用权限，如代码清单29-9所示。

代码清单29-9　添加私有权限（AndroidManifest.xml）

```
<manifest ...>

  <permission android:name="com.bignerdranch.android.photogallery.PRIVATE"
    android:protectionLevel="signature" />

  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  <uses-permission android:name="com.bignerdranch.android.photogallery.PRIVATE" />
    ...
</manifest>
```

以上代码中，你使用protection level签名定义了自己的定制权限。稍后，还会学习到更多有关protection level的知识。如同前面用过的intent操作、类别和系统权限，权限本身只是一行简单的字符串。即使是自定义的权限，也必须在使用这个权限前获取它，这是规则。

注意代码中的加亮常量，这样的字符串需要在三个地方出现，并且要保证完全一致。因此，最好使用复制粘贴功能，而不是手动输入。

现在，为使用权限，在代码中定义一个对应常量，然后将其传入`sendBroadcast(...)`方法，如代码清单29-10所示。

代码清单29-10　发送带有权限的broadcast（PollService.java）

```
public class PollService extends IntentService {
    ...
    public static final String ACTION_SHOW_NOTIFICATION =
            "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";
    public static final String PERM_PRIVATE =
            "com.bignerdranch.android.photogallery.PRIVATE";

    public static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        ...
        String resultId = items.get(0).getId();
        if (resultId.equals(lastResultId)) {
            Log.i(TAG, "Got an old result: " + resultId);
        } else {
            ...
            notificationManager.notify(0, notification);

            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
        }

        QueryPreferences.setLastResultId(this, resultId);
    }
    ...
}
```

29

要使用权限，需将其作为参数传入`sendBroadcast(...)`方法。有了这个权限，所有应用都必须使用同样的权限才能接收你发送的intent。

该如何保护你的broadcast receiver呢？其他应用可通过创建自己的broadcast intent来触发它。同样，在`registerReceiver(...)`方法中传入自定义权限就能解决该问题，如代码清单29-11所示。

代码清单29-11　broadcast receiver的使用权限（VisibleFragment.java）

```
public abstract class VisibleFragment extends Fragment {
    ...
    @Override
```

```
    public void onStart() {
        super.onStart();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter,
                PollService.PERM_PRIVATE, null);
    }
    ...
}
```

现在，只有photoGallery应用才能够触发目标receiver了。

深入学习安全级别

自定义权限必须指定`android:protectionLevel`属性值。Android根据`protectionLevel`属性值确定自定义权限的使用方式。在PhotoGallery应用中，我们使用的`protectionLevel`是`signature`。

`signature`安全级别表明，如果其他应用需要使用你的自定义权限，就必须使用和当前应用相同的key做签名认证。对于仅限应用内部使用的权限，选择`signature`安全级别比较合适。既然其他开发者没有相同的key，自然也就无法接触到权限保护的东西。此外，有了自己的key，将来还可用于你开发的其他应用中。

表29-1　protectionLevel的可选值

可　选　值	用法描述
normal	用于阻止应用执行危险操作，如访问个人隐私数据、地图定位等。应用安装前，用户可看到相应的安全级别，但用户不会被明确要求给予授权。android.permission.RECEIVE_BOOT_COMPLETED使用该安全级别。同样，应用让手机振动时，也使用该安全级别
dangerous	用于normal安全级别控制以外的任何危险操作，如访问个人隐私数据、使用可监视用户的硬件功能等。总之，包括一切可能会给用户带来麻烦的行为。网络使用权限、相机使用权限以及联系人信息使用权限都属于危险操作。Marshmallow开始，dangerous权限需要在运行时调用requestPermission(...)方法明确要求用户授权
signature	如果应用签署了与声明应用一致的权限证书，则该权限由系统授予。否则，系统会拒绝授权。权限授予时，系统不会通知用户。它通常适用于应用内部。只要拥有证书，则只有签署了同样证书的应用才能拥有该权限，因此开发者可自由控制权限的使用。前例中，它用来阻止其他应用监听PhotoGallery应用发出的broadcast。不过，如有需要，可开发能够监听它们的其他应用
signatureOrSystem	类似signature授权级别。但该授权级别针对Android系统镜像中的所有包授权。该授权级别用于系统镜像内应用间的通信。权限授予时，系统不会通知用户。开发人员一般不会用到它

29.3.4　使用有序 broadcast 收发数据

最后，你的任务是保证动态登记的receiver总是先于其他receiver接收到`PollService.ACTION_SHOW_NOTIFICATION` broadcast。然后，还要修改这个broadcast，制止通知消息的发布。

现在，虽然可以发送个人私有的broadcast了，但目前还只是发而不收的单向通信，如图29-5所示。

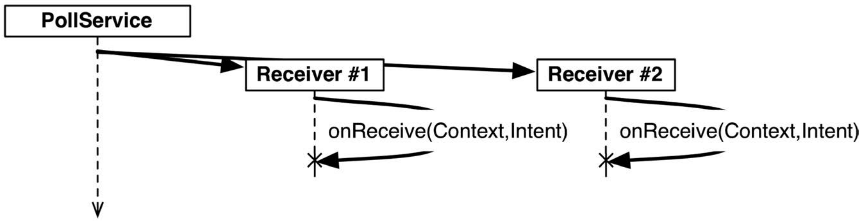


图29-5 普通broadcast intent

这是因为，普通broadcast intent只是概念上同时被所有人接收。而事实上，`onReceive(...)`方法是在主线程上调用的，所以各个receiver并没有同步并发运行。因而，不可能指望它们按照某种顺序依次运行，也不知道它们什么时候全部结束运行。结果就是，无论是broadcast receiver之间要通信，还是intent发送者要从receiver接收反馈信息，处理起来都很困难。

不过，我们可以使用有序broadcast intent来实现双向通信（如图29-6所示）。有序broadcast允许多个broadcast receiver依序处理broadcast intent。另外，通过传入一个名为result receiver的特别broadcast receiver，有序broadcast还支持让broadcast发送者接收broadcast接收者的返回结果。

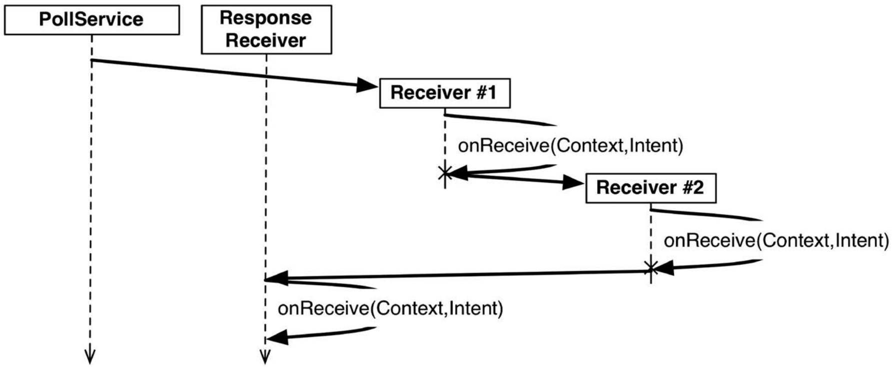


图29-6 有序broadcast intent

从接收方来看，这看上去与一般broadcast没什么不同。然而，我们却因此获得了特别的工具：一套改变receiver返回值的方法。这里，我们需要取消通知信息。这很简单，使用一个简单的整型结果码，将此要求告诉信息发送者就可以了。稍后，我们会使用`setResultCode(int)`方法设置一个`Activity. RESULT_CANCELED`结果码。

修改`VisibleFragment`类，告诉`SHOW_NOTIFICATION`的发送者应该如何处置通知消息，如代码清单29-12所示。这个信息也会发送给接收链中的所有broadcast receiver。

代码清单29-12 返回一个简单结果码（VisibleFragment.java）

```
public abstract class VisibleFragment extends Fragment {
    private static final String TAG = "VisibleFragment";
    ...
```

29

```
    private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getActivity(),
                    "Got a broadcast:" + intent.getAction(),
                    Toast.LENGTH_LONG)
                 .show();
            // If we receive this, we're visible, so cancel
            // the notification
            Log.i(TAG, "canceling notification");
            setResultCode(Activity.RESULT_CANCELED);
        }
    };
    ...
}
```

我们只需要YES或NO指示，因此使用int结果码就行。如需返回更多复杂数据，可调用setResultData(String)或setResultExtras(Bundle)方法。如需设置所有三个参数值，那就调用setResult(int, String, Bundle)方法。设定返回值后，每个后续接收者均可看到或修改它们。

为了让以上方法发挥作用，broadcast必须有序。在PollService类中，编写一个可发送有序broadcast的新方法，如代码清单29-13所示。该方法打包一个Notification调用，然后以一个broadcast发出。在onHandleIntent(...)方法中，删除原来直接发布通知给Notification-Manager的代码，调用这个新方法发出一个有序broadcast。

代码清单29-13　发送有序broadcast（PollService.java）

```
public static final String PERM_PRIVATE =
        "com.bignerdranch.android.photogallery.PRIVATE";
public static final String REQUEST_CODE = "REQUEST_CODE";
public static final String NOTIFICATION = "NOTIFICATION";
...
@Override
protected void onHandleIntent(Intent intent) {
    ...
    String resultId = items.get(0).getId();
    if (resultId.equals(lastResultId)) {
        Log.i(TAG, "Got an old result: " + resultId);
    } else {
        Log.i(TAG, "Got a new result: " + resultId);
        ...

        Notification notification = ...;

        NotificationManagerCompat notificationManager =
                NotificationManagerCompat.from(this);
        notificationManager.notify(0, notification);

        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
        showBackgroundNotification(0, notification);
    }
```

```
        QueryPreferences.setLastResultId(this, resultId);
    }

    private void showBackgroundNotification(int requestCode, Notification notification) {
        Intent i = new Intent(ACTION_SHOW_NOTIFICATION);
        i.putExtra(REQUEST_CODE, requestCode);
        i.putExtra(NOTIFICATION, notification);
        sendOrderedBroadcast(i, PERM_PRIVATE, null, null,
                Activity.RESULT_OK, null, null);
    }
```

除了在sendBroadcast(Intent,String)方法中使用的两个参数外，Context.sendOrderedBroadcast(Intent, String, BroadcastReceiver, Handler, int, String, Bundle)方法还有另外五个参数，依次为：一个result receiver，一个支持result receiver运行的Handler，结果代码初始值，结果数据以及有序broadcast的结果附加内容。

result receiver比较特殊，只有在所有有序broadcast intent接收者结束运行后，它才开始运行。虽然有时也能使用result receiver接收broadcast和发布通知对象，但此处行不通。目标broadcast intent通常是在PollService对象消亡之前发出的，这意味着broadcast receiver很可能也不存在了。

因此，最终的broadcast receiver应该是个standalone receiver。而且，无论如何都要保证它在其他动态登记的receiver之后运行。

首先，新建一个NotificationReceiver类，继承BroadcastReceiver类，如代码清单29-14所示。

代码清单29-14　实现result receiver（NotificationReceiver.java）

```
public class NotificationReceiver extends BroadcastReceiver {
    private static final String TAG = "NotificationReceiver";

    @Override
    public void onReceive(Context c, Intent i) {
        Log.i(TAG, "received result: " + getResultCode());
        if (getResultCode() != Activity.RESULT_OK) {
            // A foreground activity cancelled the broadcast
            return;
        }

        int requestCode = i.getIntExtra(PollService.REQUEST_CODE, 0);
        Notification notification = (Notification)
                i.getParcelableExtra(PollService.NOTIFICATION);

        NotificationManagerCompat notificationManager =
                NotificationManagerCompat.from(c);
        notificationManager.notify(requestCode, notification);
    }
}
```

29

然后，登记这个新建的receiver并赋予它优先级。为保证NotificationReceiver最后一个接收目标broadcast（这样，它就知道该不该向NotificationManager发出通知），需要为它设置一

个低优先级。要让它最后一个运行，设置其优先级值为–999，这是用户能定义的最低优先级（–1000及以下值是系统保留值）。

另外，既然NotificationReceiver仅限PhotoGallery应用内部使用，还需设置一个android:exported="false"属性值，以确证外部应用看不到它，如代码清单29-15所示。

代码清单29-15　登记notification receiver（AndroidManifest.xml）

```
<receiver android:name=".StartupReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
<receiver android:name=".NotificationReceiver"
          android:exported="false">
    <intent-filter android:priority="-999">
        <action
            android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION" />
    </intent-filter>
</receiver>
```

运行PhotoGallery应用，多次切换后台polling状态。可以看到，应用开着的时候，通知信息不会出现了。（为避免傻等15分钟，可再次将PollService.POLL_INTERVAL_MS的时间间隔设置为60秒。）

29.4　receiver 与长时运行任务

如不愿受制于主线程，希望用broadcast intent触发一个长时运行任务，该怎么做呢？

有两种方法可以选择。第一种方法是将任务交给服务去处理，然后通过broadcast receiver瞬时启动服务。这是我们首推的方式。服务可以一直运行，直到完成要处理的任务。服务还能将请求放在队列中，然后依次处理，或按其自认为合适的方式管理全部请求。

第二种方法是使用BroadcastReceiver.goAsync()。该方法返回一个BroadcastReceiver.PendingResult对象，随后可使用该对象提供结果。因此，可将PendingResult交给一个AsyncTask去执行长时任务，然后再调用PendingResult的方法响应broadcast。

goAsync()方法的弊端是不够灵活。我们仍需快速响应broadcast（10秒内），并且与使用服务相比，没多少架构模式好选了。

当然，goAsync()方法也有个明显的优势：可调用该方法设置有序broadcast的结果。如果别无选择，真的要使用，应注意控制尽快完事。

29.5　深入学习：本地事件

broadcast intent可实现系统内全局性的消息传递。如果仅需要应用进程内的消息事件广播，该怎么做呢？答案是使用事件总线（event bus）。

事件总线的设计思路就是，提供一个应用内的部件可以订阅的共享总线或数据流。事件一旦

发布到总线上，各订阅部件就会被激活并执行相应的回调代码。

greenrobot出品的EventBus是目前广为人知的一个第三方事件总线库。其他常用的事件总线库还有Square的Otto。或者也可以使用RxJava `Subject`和`Observable`来模拟事件总线。

为实现在应用内发送broadcast intent，Android自己也提供了一个叫作`LocalBroadcast-Manager`的广播管理类。不过，两相比较，还是上述第三方类库用起来更为灵活和方便。

29.5.1 使用 EventBus

要在应用中使用EventBus，首先需要在项目中添加依赖库。然后，就可以定义事件类了（如果需要传送数据，可以向事件里添加数据字段）：

```
public class NewFriendAddedEvent { }
```

在应用的任何地方，都可以把消息事件发布到总线上：

```
EventBus eventBus = EventBus.getDefault();
eventBus.post(new NewFriendAddedEvent());
```

在总线上登记监听，应用的其他部分也可以订阅接收事件消息。通常，activity或fragment的登记和撤销登记都是在相应的生命周期方法中处理的，如`onStart(...)`和`onStop(...)`。

```
// In some fragment or activity...
private EventBus mEventBus;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mEventBus = EventBus.getDefault();
}

@Override
public void onStart() {
    super.onStart();
    mEventBus.register(this);
}

@Override
public void onStop() {
    super.onStop();
    mEventBus.unregister(this);
}
```

有订阅的事件消息发布时，可实施一个方法，传入合适的事件类型并添加`@Subscribe`注解，让订阅者作出响应。如果使用不带参数的`@Subscribe`注解，事件消息来自哪个线程，就在哪个线程上处理。如果使用`@Subscribe(threadMode = ThreadMode.MAIN)`，可确保事件在主线程上处理，哪怕它碰巧来自后台线程。

```
// In some registered component, like a fragment or activity...
@Subscribe(threadMode = ThreadMode.MAIN)
public void onNewFriend Added(NewFriendAddedEvent event){
    Friend newFriend = event.getFriend();
```

```
        // Update the UI or do something in response to event...
    }
```

29.5.2　使用 RxJava

RxJava也能用来实现事件广播机制。RxJava库可用来开发reactive风格的Java代码。上述reactive概念有深广的含义，不在本书讨论之列。简而言之，有了RxJava，就可以发布和订阅各类事件，并且还有很多通用工具用来管理这些事件。

所以，你可以创建一个叫Subject的对象，然后发布事件给它以及在其上订阅事件。

```
Subject<Object, Object> eventBus = new SerializedSubject<>(PublishSubject.create());
```

可以像这样发布事件给它：

```
Friend someNewFriend = ...;
NewFriendAddedEvent event = new NewFriendAddedEvent(someNewFriend);
eventBus.onNext(event);
```

并且在其上订阅事件：

```
eventBus.subscribe(new Action1<Object>() {
    @Override
    public void call(Object event) {
        if (event instanceof NewFriendAddedEvent) {
            Friend newFriend = ((NewFriendAddedEvent)event).getFriend();
            // Update the UI
        }
    }
})
```

RxJava解决方案的优势在于，eventBus现在也是个Observable对象（代表RxJava的事件流）了。这就意味着RxJava的各种事件管理工具都可以为你所用了。是不是愈发感兴趣了？去看看RxJava的项目wiki主页吧：github.com/ReactiveX/RxJava/wiki 。

29.6　深入学习：探测 fragment 的状态

本章，在实现PhotoGallery应用的通知功能时，我们使用了全局性的broadcast机制。broadcast虽然是全局的，但利用定制权限，我们实现限定broadcast intent只能在应用内接收。这就不免让人疑惑：“既然要限制，为什么还要使用全局机制？使用本地broadcast机制不是更好吗？”

这是因为有个难题要解决：如何判断PhotoGalleryFragment的活动状态。最终，利用有序broadcast、standalone receiver以及动态登记的receiver，问题总算解决了。虽然没那么干净利落，但这是Android目前能提供的最好解决方案。

更具体来讲，不用本地broadcast机制，就是因为LocalBroadcastManager既无法处理PhotoGallery应用里的这种broadcast通知，也无法知晓fragment的状态。如果继续深究，原因不外乎两点。

首先，LocalBroadcastManager不支持有序broadcast（虽然它有个sendBroadcastSync

(Intent intent)方法，但依然不灵），而在PhotoGallery应用中，不使用有序broadcast，就无法控制NotificationReceiver最后一个运行。

其次，sendBroadcastSync(Intent intent)方法不支持在独立线程上发送和接收broadcast。而在PhotoGallery应用中，需要在后台线程上发送broadcast（使用PollService.onHandleIntent(...)方法），在主线程上接收intent（在主线程上的onStart(...)方法中，使用PhotoGalleryFragment登记的动态receiver）。

撰写本书时，关于LocalBroadcastManager究竟是如何处理线程上的broadcast投递的，Android还没有确切的说明。但经验告诉我们，它是有规律可循的。例如，如果从后台线程调用sendBroadcastSync(...)，所有的pending broadcast都会在后台线程上涌出，不管是不是来自主线程。

当然，LocalBroadcastManager也不是一无是处，只不过不适合解决本章的问题而已。

第 30 章 网页浏览

从Flickr下载的图片都有其关联网页。本章，我们继续升级PhotoGallery应用，让用户点击图片就能看到它的Flickr网页。我们会以两种不同的方式整合网页内容，完成后的效果如图30-1所示：左边是使用浏览器应用，右边是使用WebView在应用中显示网页内容。

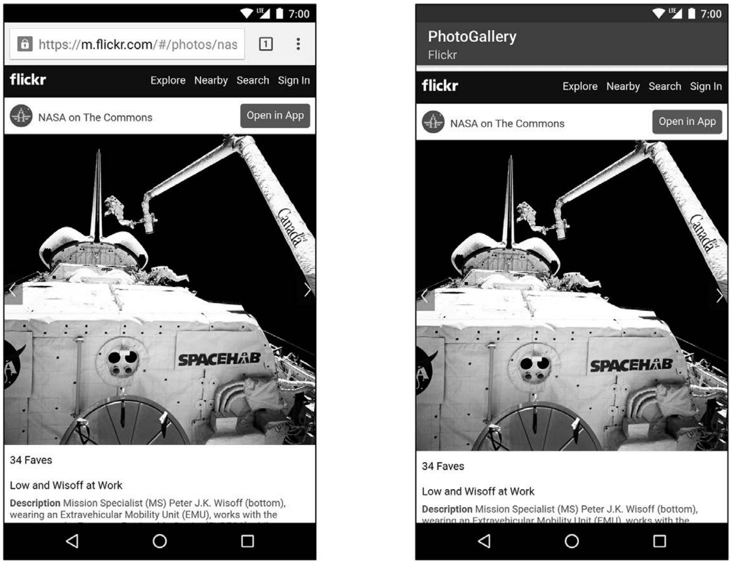


图30-1　以两种方式呈现Web内容

30.1　最后一段 Flickr 数据

无论哪种方式，都需要取得图片的Flickr网页URL。如果查看下载图片的JSON文件，可看到图片的网页地址并不包含在内。

```
{
  "photos": {
    ...,
    "photo": [
```

```
        {
          "id": "9452133594",
          "owner": "44494372@N05",
          "secret": "d6d20af93e",
          "server": "7365",
          "farm": 8,
          "title": "Low and Wisoff at Work",
          "ispublic": 1,
          "isfriend": 0,
          "isfamily": 0,
          "url_s":"https://farm8.staticflickr.com/7365/9452133594_d6d20af93e_m.jpg"
        }, ...
      ]
    },
    "stat": "ok"
}
```


因此，你想当然地认为需要编码获取更多JSON内容才行。实际上，并不是这样的。访问www.flickr.com/services/api/misc.urls.html查看Flickr官方文档的Web Page URLs部分可知，可按以下格式创建单个图片的URL：

```
http://www.flickr.com/photos/user-id/photo-id
```

这里的`photo-id`即JSON数据里的`id`属性值。该值已保存在`GalleryItem`类的`mId`属性中。那么`user-id`呢？继续查阅Flickr文档可知，JSON文件的`owner`属性值就是用户ID。因此，只需从JSON文件解析出`owner`属性值，即可创建图片的完整URL：

```
http://www.flickr.com/photos/owner/id
```

在`GalleryItem`中添加代码清单30-1所示代码，创建图片URL。

代码清单30-1　添加创建图片URL的代码（GalleryItem.java）

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;
    private String mOwner;
    ...
    public void setUrl(String url) {
        mUrl = url;
    }

    public String getOwner() {
        return mOwner;
    }

    public void setOwner(String owner) {
        mOwner = owner;
    }

    public Uri getPhotoPageUri() {
        return Uri.parse("http://www.flickr.com/photos/")
```

```
                .buildUpon()
                .appendPath(mOwner)
                .appendPath(mId)
                .build();
    }

    @Override
    public String toString() {
        return mCaption;
    }
}
```

以上代码新建了一个mOwner属性，以及一个产生图片URL的getPhotoPageUri()方法。

现在，修改parseItems(...)方法，从JSON数据中获取owner属性，如代码清单30-2所示。

代码清单30-2　从JSON数据中获取owner属性（FlickrFetchr.java）

```
public class FlickrFetchr {
    ...
    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
            throws IOException, JSONException {

        JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
        JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

        for (int i = 0; i < photoJsonArray.length(); i++) {
            JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

            GalleryItem item = new GalleryItem();
            item.setId(photoJsonObject.getString("id"));
            item.setCaption(photoJsonObject.getString("title"));

            if (!photoJsonObject.has("url_s")) {
                continue;
            }

            item.setUrl(photoJsonObject.getString("url_s"));
            item.setOwner(photoJsonObject.getString("owner"));
            items.add(item);
        }
    }
}
```

非常简单，获取图片网页URL的任务就完成了。

30.2　简单方式：隐式 intent

首先使用隐式intent这个老朋友来访问图片URL。隐式intent可启动浏览器，并在其中打开图片URL指向的网页。

首先，监听RecyclerView显示项的点击事件。更新PhotoGalleryFragment类的PhotoHolder，实现一个可以发送隐式intent的事件监听方法，如代码清单30-3所示。

代码清单30-3　通过隐式intent实现网页浏览（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends VisibleFragment {
    ...
    private class PhotoHolder extends RecyclerView.ViewHolder
             implements View.OnClickListener {
        private ImageView mItemImageView;
        private GalleryItem mGalleryItem;

        public PhotoHolder(View itemView) {
            super(itemView);

            mItemImageView = (ImageView) itemView.findViewById(R.id.item_image_view);
            itemView.setOnClickListener(this);
        }

        public void bindDrawable(Drawable drawable) {
            mItemImageView.setImageDrawable(drawable);
        }

        public void bindGalleryItem(GalleryItem galleryItem) {
            mGalleryItem = galleryItem;
        }

        @Override
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_VIEW, mGalleryItem.getPhotoPageUri());
            startActivity(i);
        }
    }
    ...
}
```

然后，在`PhotoAdapter.onBindViewHolder(...)`方法中绑定`PhotoHolder`给`GalleryItem`，如代码清单30-4所示。

30

代码清单30-4　绑定`GalleryItem`（PhotoGalleryFragment.java）

```
private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
    ...
    @Override
    public void onBindViewHolder(PhotoHolder photoHolder, int position) {
        GalleryItem galleryItem = mGalleryItems.get(position);
        photoHolder.bindGalleryItem(galleryItem);
        Drawable placeholder = getResources().getDrawable(R.drawable.bill_up_close);
        photoHolder.bindDrawable(placeholder);
        mThumbnailDownloader.queueThumbnail(photoHolder, galleryItem.getUrl());
    }
    ...
}
```

搞定了。启动PhotoGallery应用并点击任意图片。浏览器应用应该会弹出并加载显示对应的图片网页（类似图30-1的左边）。

30.3　较难方式：使用 WebView

使用隐式intent打开图片网页简单又高效。但是，如果不想打开独立的浏览器怎么办？

通常，我们只想在activity中显示网页内容，而不是打开浏览器。这么做或许是想显示自己生成的HTML，或许是想以某种方式限制用户使用浏览器。对于大多数需要帮助文档的应用，常见的做法就是以网页的形式提供帮助文档，这样会方便后期的更新与维护。打开浏览器查看帮助文档，既不专业，又妨碍应用行为的定制，无法将网页整合进自己的用户界面。

如果想实现在应用里显示网页内容，你可以使用WebView类。这就是我们说的较难方式，但实际也没那么难。（当然，相对隐式intent来说，要困难一些。）

首先，创建一个activity以及一个显示WebView的fragment。依惯例先定义一个名为fragment_photo_page.xml的布局文件。使用一个ConstraintLayout作为一级组件。在布局编辑窗口，安排一个WebView作为ConstraintLayout的子组件。（WebView位于Containers区的下面。）

添加完WebView，相对其父组件，为每一边添加一个约束。具体如下：

- 从WebView顶部到其父组件顶部
- 从WebView底部到其父组件底部
- 从WebView左边到其父组件左边
- 从WebView右边到其父组件右边

最后，高和宽设置为Any Size并设置所有的margin为0。另外，别忘了给WebView一个ID：web_view。

是不是认为这里的ConstraintLayout没什么用？现在是这样，稍后会添加更多组件来完善它。

接下来创建fragment。新建PhotoPageFragment类，继承上一章的VisibleFragment类。然后，在这个新类中，实例化布局文件，引用WebView，并转发从intent数据中获取的URL，如代码清单30-5所示。

代码清单30-5　创建网页浏览fragment（PhotoPageFragment.java）

```
public class PhotoPageFragment extends VisibleFragment {
    private static final String ARG_URI = "photo_page_url";

    private Uri mUri;
    private WebView mWebView;

    public static PhotoPageFragment newInstance(Uri uri) {
        Bundle args = new Bundle();
        args.putParcelable(ARG_URI, uri);

        PhotoPageFragment fragment = new PhotoPageFragment();
        fragment.setArguments(args);
        return fragment;
    }

    @Override
```

```
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            mUri = getArguments().getParcelable(ARG_URI);
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                 Bundle savedInstanceState) {
            View v = inflater.inflate(R.layout.fragment_photo_page, container, false);

            mWebView = (WebView) v.findViewById(R.id.web_view);

            return v;
        }
    }
```

当前，PhotoPageFragment类还未完成，稍后再来完成它。接下来，新建PhotoPageActivity托管类，继承SingleFragmentActivity类，如代码清单30-6所示。

代码清单30-6　创建显示网页的activity（PhotoPageActivity.java）

```
public class PhotoPageActivity extends SingleFragmentActivity {

    public static Intent newIntent(Context context, Uri photoPageUri) {
        Intent i = new Intent(context, PhotoPageActivity.class);
        i.setData(photoPageUri);
        return i;
    }

    @Override
    protected Fragment createFragment() {
        return PhotoPageFragment.newInstance(getIntent().getData());
    }
}
```


回到PhotoGalleryFragment类中，弃用隐式intent，启动新建的activity，如代码清单30-7所示。

代码清单30-7　启动新建的activity（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends VisibleFragment {
    ...
    private class PhotoHolder extends RecyclerView.ViewHolder
            implements View.OnClickListener{
        ...
        @Override
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_VIEW, mGalleryItem.getPhotoPageUri());
            Intent i = PhotoPageActivity
                .newIntent(getActivity(), mGalleryItem.getPhotoPageUri());
            startActivity(i);
        }
    }
```

```
    ...
}
```

最后，在配置文件中声明新建的activity，如代码清单30-8所示。

代码清单30-8　在配置文件中声明activity（AndroidManifest.xml）

```
<manifest ... >
    ...
    <application
        ...>
        <activity
            android:name=".PhotoGalleryActivity"
            android:label="@string/app_name" >
            ...
        </activity>

        <activity
            android:name=".PhotoPageActivity" />

        <service android:name=".PollService" />
        ...
    </application>

</manifest>
```

运行PhotoGallery应用，点击任意图片，可看到一个新的空activity弹出。

好了，现在来处理关键部分，让fragment发挥其作用。WebView要显示Flickr图片网页，需做三件事。

首先是告诉WebView要打开的URL。

其次是启用JavaScript。JavaScript默认是禁用的。虽然并不总是需要启用它，但Flickr网站需要。（启用JavaScript后，Android Lint会提示警告信息（担心跨网站的脚本攻击），可以使用@SuppressLint("SetJavaScriptEnabled")注解onCreateView(...)方法以禁止Lint的警告。）

最后，需要实现一个WebViewClient类（用来响应WebView上的渲染事件）。添加代码清单30-9所示代码。然后，我们来详细解读PhotoPageFragment类。

代码清单30-9　加载URL（PhotoPageFragment.java）

```
public class PhotoPageFragment extends VisibleFragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, container, false);

        mWebView = (WebView) v.findViewById(R.id.web_view);
        mWebView.getSettings().setJavaScriptEnabled(true);
        mWebView.setWebViewClient(new WebViewClient() {
        mWebView.loadUrl(mUri.toString());

        return v;
```

```
        }
    }
```

加载URL必须等WebView配置完成后进行，因此最后再执行这一操作。在此之前，首先调用getSettings()方法获得WebSettings实例，再调用WebSettings.setJavaScriptEnabled(true)方法启用JavaScript。WebSettings是修改WebView配置的三种途径之一。另外还有其他一些可设置属性，如用户代理字符串和显示文字大小。

然后，添加一个WebViewClient给WebView。为什么要添加？解释之前，我们先看看没有它会发生什么。

载入一个新URL有好几种方式：当前页面刷新让你转入另一个URL（一个重定向），或者点击一个链接载入。如果没有WebViewClient，WebView会要求activity管理器找一个新activity来载入新URL。

这不是我们想要的。如果从手机浏览器加载，许多网址（包括Flickr图片网页）会重定向到移动版本的网址。因此，发送一个隐式intent启动其他浏览器应用解决不了问题。我们需要在自己应用里展示网页。

所以，给WebView添加一个WebViewClient，事情就不一样了。现在，WebView不会再去麻烦activity管理器，它会去找WebViewClient。按照WebViewClient的默认实现，它会说："WebView，自己载入URL吧。"这样，目标网页就在WebView中打开了。

运行应用，点击任意图片，应该可以看到显示对应图片的WebView（类似图30-1的右边）。

使用 WebChromeClient 优化 WebView 显示

既然花时间实现了自己的WebView，接下来开始优化，为它添加一个标题视图和一个进度条。以视图的方式打开fragment_photo_page.xml，拖入一个ProgressBar组件作为ConstraintLayout的第二个子组件（使用ProgressBar（Horizontal）版本））。删除WebView最上面的约束。然后为方便使用它的约束handle，设置其高度值为Fixed。

30

完成后，再创建以下额外约束。

- 从ProgressBar到其父组件的上，左，右。
- 从WebView的顶部到ProgressBar的底部。

接下来，重置WebView的高度为Any Size，设置ProgressBar的高度为wrap_content，宽度为Any Size。

最后，选中ProgressBar组件，在右边的属性窗口，将visibility和tools visibility分别设置为gone和visible，最后重命名其ID为progress_bar。

完成后的结果图如图30-2所示。

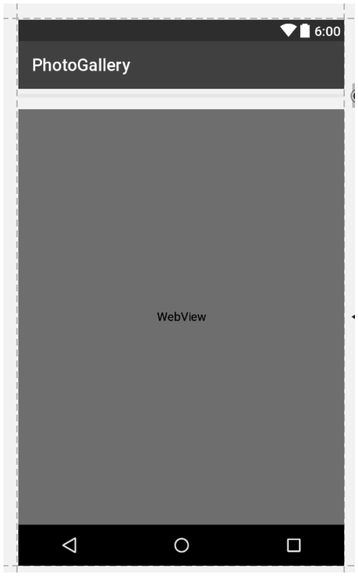


图30-2　添加ProgressBar

为使用ProgressBar，还需使用WebView:WebChromeClient的第二个回调方法。如果说WebViewClient是响应渲染事件的接口，那么WebChromeClient就是一个事件接口，用来响应那些改变浏览器中装饰元素的事件。这包括JavaScript警告信息、网页图标、状态条加载，以及当前网页标题的刷新。

在onCreateView(...)方法中，编码实现使用WebChromeClient，如代码清单30-10所示。

代码清单30-10　使用WebChromeClient（PhotoPageFragment.java）

```
public class PhotoPageFragment extends VisibleFragment {
    ...
    private WebView mWebView;
    private ProgressBar mProgressBar;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, container, false);

        mProgressBar = (ProgressBar)v.findViewById(R.id.progress_bar);
        mProgressBar.setMax(100); // WebChromeClient reports in range 0-100


        mWebView = (WebView) v.findViewById(R.id.web_view);
        mWebView.getSettings().setJavaScriptEnabled(true);
        mWebView.setWebChromeClient(new WebChromeClient() {
            public void onProgressChanged(WebView webView, int newProgress) {
                if (newProgress == 100) {
                    mProgressBar.setVisibility(View.GONE);
```

```
                } else {
                    mProgressBar.setVisibility(View.VISIBLE);
                    mProgressBar.setProgress(newProgress);
                }
            }

            public void onReceivedTitle(WebView webView, String title) {
                AppCompatActivity activity = (AppCompatActivity) getActivity();
                activity.getSupportActionBar().setSubtitle(title);
            }
        });
        mWebView.setWebViewClient(new WebViewClient());
        mWebView.loadUrl(mUri.toString());

        return v;
    }
}
```

进度条和标题栏更新都有各自的回调方法，即onProgressChanged(WebView, int)和onReceivedTitle(WebView, String)方法。从onProgressChanged(WebView, int)方法收到的网页加载进度是一个从0到100的整数值。如果值是100，说明网页已完成加载，因此需设置进度条可见性为View.GONE，将ProgressBar视图隐藏起来。

运行PhotoGallery应用。现在，应看到类似图30-3的应用画面。



图30-3　漂亮的WebView

点击任意图片，PhotoPageActivity会出现。网页加载时，会出现进度条，工具栏会出现来自onReceivedTitle(...)方法的子标题。页面加载完毕，进度条随即消失。

30.4　处理 WebView 的设备旋转问题

尝试旋转设备屏幕。尽管应用工作如常，但WebView重新加载了网页。这是因为WebView包含太多的数据，无法在onSaveInstanceState(...)方法内全部保存。所以每次设备旋转，它都必须从头开始加载网页数据。

是不是想到了PhotoPageFragment保留？不好意思，这里行不通。因为WebView是视图层级结构的一部分，所以旋转后它肯定会销毁并重建。

对于一些类似的类（如VideoView），Android文档推荐让activity自己处理设备配置变更。也就是说，无需销毁重建activity，就能直接调整自己的视图以适应新的屏幕尺寸。这样，WebView也就不必重新加载全部数据了。

为了让PhotoPageActivity自己处理设备配置调整，可在AndroidManifest.xml配置文件中做如下调整，如代码清单30-11所示。

代码清单30-11　自己处理设备配置更改（AndroidManifest.xml）

```
<manifest ... >
    ...
    <activity
          android:name=".PhotoPageActivity"
          android:configChanges="keyboardHidden|orientation|screenSize" />
      ...
</manifest>
```

android:configChanges属性表明，如果因键盘开或关、屏幕方向改变、屏幕大小改变（也包括Android 3.2之后的屏幕方向变化）而发生设备配置更改，那么activity应自己处理配置更改。

运行应用，再次尝试旋转设备，一切都完美了。

自己处理配置更改的风险

自己处理设备配置更改，我们轻松搞定了WebView的设备旋转问题。既然这么简单，为什么不全面推广使用这个方法呢？实际上，事情没那么简单，自己处理配置变更也是有风险的。

首先，资源修饰符无法自动工作了。开发人员必须手工重载视图。这实际是非常棘手的。

其次，也是更重要的一点，既然activity自己处理配置更改了，你很可能不会去覆盖Activity.onSavedInstanceState(...)方法存储UI状态。然而，这依然是必需的，即使自己处理设备配置更改也是一样。因为低内存情况还是要考虑的。（还记得吗？activity不运行的时候，系统可能会销毁并暂存它的状态，如第3章图3-14中看到的那样。）

30.5　深入学习：注入 JavaScript 对象

你已经知道如何使用WebViewClient和WebChromeClient类响应发生在WebView里的特定事件。然而，通过注入任意JavaScript对象到WebView本身包含的文档中，还可以做到更多。打开文档网页developer.android.com/reference/android/webkit/WebView.html，找到addJavascript-

`Interface(Object, String)`方法看看。使用该方法，可注入任意JavaScript对象到指定文档中：

```
mWebView.addJavascriptInterface(new Object() {
    @JavascriptInterface
    public void send(String message) {
        Log.i(TAG, "Received message: " + message);
    }
}, "androidObject");
```

然后按如下方式调用：

```
<input type="button" value="In WebView!"
    onClick="sendToAndroid('In Android land')" />

<script type="text/javascript">
    function sendToAndroid(message) {
        androidObject.send(message);
    }
</script>
```

上述代码有些地方值得一说。首先，调用`send(String)`方法时，它不是在主线程上，而是在WebView拥有的线程上被调的。所以，要是有Android UI更新任务，你需要使用Handler把控制传回给主线程。

其次，除了String，其他好多数据类型都没法支持。如果有其他复杂数据类型，那只能转成String类型，比如，转成常见的JSON格式。使用者收到后，自己再去按需解析。

自Jelly Bean 4.2（API 17）开始，只有以`@JavascriptInterface`注解的公共方法才会暴露给JavaScrpit。在这之前，所有对象树中的公共方法都是开放访问的。

这可能有风险，因为一些可能的问题网页能够直接接触到应用。安全起见，要么自己掌控局面，要么严格控制不要暴露自己的接口。

30.6 深入学习：WebView 升级

基于Chromium开源项目，随KitKat 4.4（API 19）发布的`WebView`经历了一次大修。新`WebView`使用了和Android版Chrome一样的渲染引擎。现在，两者的页面外观和浏览器行为越来越趋于统一了。（然而，`WebView`并不具有Android Chrome的全部特性。查看developer.chrome.com/multidevice/webview/overview，可看到它们的对照表。）

转向Chromium给`WebView`带来了一系列激动人心的改进，比如支持HTML5和CSS3新网页标准，一个全新的JavaScript引擎以及增强的网页展示性能。从开发者的角度看，一个最令人兴奋的新特性就是，`WebView`终于支持使用Chrome DevTools进行远程调试了（调用`WebView.setWebContentsDebuggingEnabled()`方法开启）。

自Lollipop（Android 5.0）开始，`WebView`的Chromium层会自动从Google Play商店更新。等Android发布新系统版本才能升级安全补丁或用上新功能的日子终于熬到头了。对开发者来说，这是个大事件。

现在，对于Nougat（Android 7.0），`WebView`的Chromium层又改为直接来自Chrome APK安装

包。这变化也太快了。没事，无论如何，只要WebView能一直保持更新，就该满足了。

30.7　挑战练习：使用后退键浏览历史网页

注意到了没有，启动PhotoPageActivity之后，还可以在WebView中点击跳转到其他链接。然而，不管如何跳转，访问了多少个网页，只要按后退键，就会立即回到PhotoPageActivity。如果想使用后退键在WebView里层层退回到已浏览的历史网页呢？

提示：首先覆盖后退键方法Activity.onBackPressed()。在该方法内，再搭配使用WebView的历史记录浏览方法（WebView.canGoBack()和WebView.goBack()）实现想要的浏览逻辑。如果WebView里有历史浏览记录，就回到前一个历史网页，否则调用super.onBackPressed()方法回到PhotoPageActivity。

30.8　挑战练习：非 HTTP 链接支持

使用PhotoPageFragment的WebView时，你可能会遇到非HTTP链接。例如，本书写作时，Flickr图片明细页会显示一个Open in App按钮。如果点击它，应该会启动Flickr应用（已安装）；没安装的话，会启动Google Play应用商店让用户选择安装它。

然而，实际点击Open in App按钮，WebView却显示了如图30-4所示的画面。

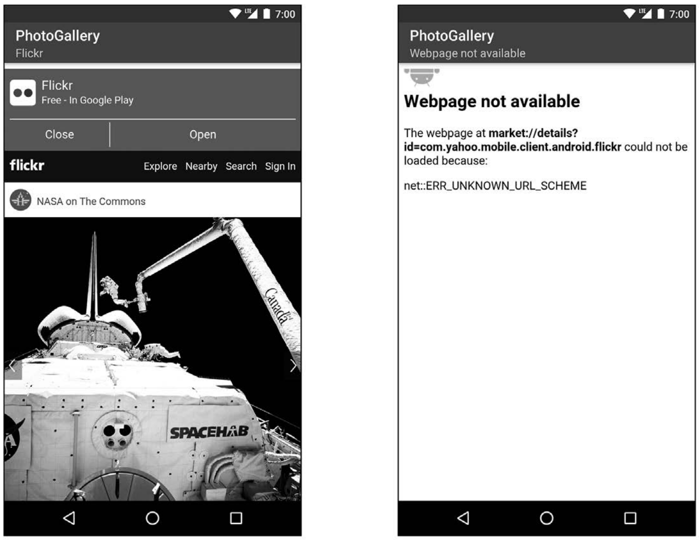


图30-4　Open in App错误

这是因为WebViewClient总是让WebView尝试自己加载URI，即使是它不支持的URI scheme。

要解决这个问题，非HTTP URI就要交给最合适的应用去处理。因此，加载URI前，先检查它的scheme，如果不是HTTP或HTTPS，就发送一个针对目标URI的Intent.ACTION_VIEW。

第 31 章 定制视图与触摸事件

本章，通过开发一个名为`BoxDrawingView`的定制`View`子类，我们来学习如何处理触摸事件。在新项目DragAndDraw中，这个定制`View`会响应用户的触摸与拖动，在屏幕上绘制出矩形框，如图31-1所示。

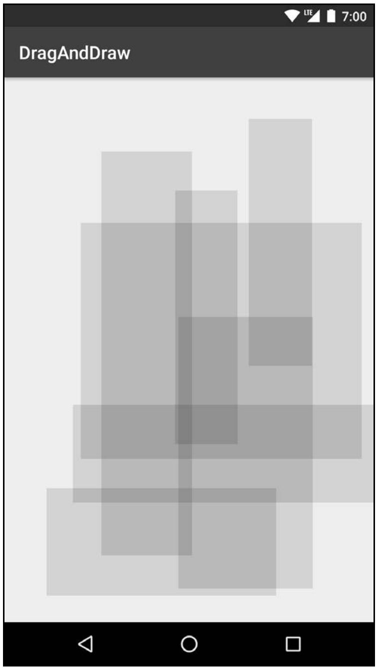


图31-1　各种形状大小的绘制框

31.1　创建 DragAndDraw 项目

创建DragAndDraw新项目，最低SDK版本选择API 19，新建空activity并命名为`DragAndDrawActivity`。

既然SingleFragmentActivity可实例化仅包含单个fragment的布局，我们让DragAndDrawActivity继承它。在Android Studio中，复制之前项目的SingleFragmentActivity.java和activity_fragment.xml文件到DragAndDraw项目的对应目录中。

在DragAndDrawActivity.java中，修改代码继承SingleFragmentActivity类，并创建返回一个DragAndDrawFragment对象（稍后会创建该类），如代码清单31-1所示。

代码清单31-1　修改activity（DragAndDrawActivity.java）

```
public class DragAndDrawActivity extends AppCompatActivity SingleFragmentActivity {

    @Override
    public Fragment createFragment() {
        return DragAndDrawFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
}
```

为准备DragAndDrawFragment的布局，重命名activity_drag_and_draw.xml为fragment_drag_and_draw.xml。

DragAndDrawFragment的布局最终由BoxDrawingView定制视图组成。稍后我们会创建这个定制视图类。它会处理所有的图形绘制和触摸事件。

继承android.support.v4.app.Fragment超类，新建DragAndDrawFragment类。然后覆盖onCreateView(...)方法，并在其中实例化fragment_drag_and_draw.xml布局，如代码清单31-2所示。

代码清单31-2　创建DragAndDrawFragment（DragAndDrawFragment.java）

```
public class DragAndDrawFragment extends Fragment {

    public static DragAndDrawFragment newInstance() {
        return new DragAndDrawFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_drag_and_draw, container, false);
        return v;
    }
}
```

运行DragAndDraw应用，确认运行效果如图31-2所示。

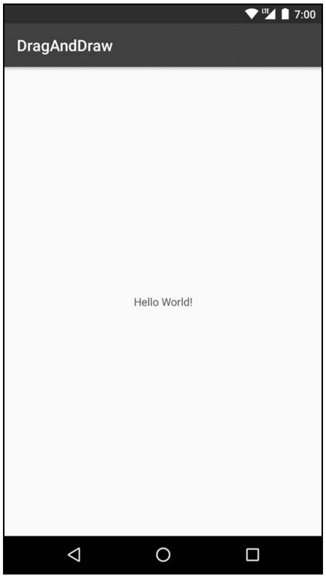


图31-2　带默认布局的DragAndDraw应用

31.2　创建定制视图

Android为开发者准备了很多标准视图与组件，但有时为追求独特的应用视觉效果，创建定制视图不可避免。

虽然定制视图很多，但总体归为以下两大类别。

- 简单视图。简单视图内部也可以很复杂，之所以归为简单类别，是因为简单视图不包括子视图。简单视图几乎总是用来处理定制绘制。
- 聚合视图。聚合视图由其他视图对象组成。聚合视图通常用来管理子视图，但不负责处理定制绘制。图形绘制任务都委托给了各个子视图。

31

以下为创建定制视图所需的三大步骤。

(1) 选择超类。对于简单定制视图而言，`View`是个空白画布，因此它作为超类最常见。对于聚合定制视图，我们应选择合适的超类布局，比如`FrameLayout`。

(2) 继承选定的超类，覆盖超类的构造方法。

(3) 覆盖其他关键方法，以定制视图行为。

创建 BoxDrawingView 视图

`BoxDrawingView`是个简单视图，同时也是`View`的直接子类。

以`View`为超类，新建`BoxDrawingView`类。在BoxDrawingView.java中，添加两个构造方法。如代码清单31-3所示。

代码清单31-3　初始的BoxDrawingView视图类（BoxDrawingView.java）

```
public class BoxDrawingView extends View {

    // Used when creating the view in code
    public BoxDrawingView(Context context) {
        this(context, null);
    }

    // Used when inflating the view from XML
    public BoxDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
}
```

这里之所以添加了两个构造方法，是因为视图可从代码或者布局文件实例化。从布局文件中实例化的视图会收到一个AttributeSet实例，该实例包含了XML布局文件中指定的XML属性。即使不打算使用构造方法，按习惯做法也应添加这两个构造方法。

有了定制视图类，我们来更新fragment_drag_and_draw.xml布局文件以使用它，如代码清单31-4所示。

代码清单31-4　在布局中添加BoxDrawingView（fragment_drag_and_draw.xml）

```
<android.support.constraint.ConstraintLayout
    android:id="@+id/activity_drag_and_draw"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.bignerdranch.android.draganddraw.DragAndDrawActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="@+id/activity_drag_and_draw"
        app:layout_constraintLeft_toLeftOf="@+id/activity_drag_and_draw"
        app:layout_constraintRight_toRightOf="@+id/activity_drag_and_draw"
        app:layout_constraintTop_toTopOf="@+id/activity_drag_and_draw"/>

</android.support.constraint.ConstraintLayout>
<com.bignerdranch.android.draganddraw.BoxDrawingView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

注意，应给出BoxDrawingView的全路径类名，这样布局inflater才能够找到它。布局inflater解析布局XML文件，并按视图定义创建View实例。如果不给全路径类名，布局inflater会转而在android.view和android.widget包中寻找同名类。显然，如果目标视图类在其他包里，布局inflater就找不到，应用就会崩溃。

所以，对于android.view和android.widget包以外的定制视图类，必须指定它们的全路径类名。

运行DragAndDraw应用，一切正常的话，屏幕上会出现一个空视图，如图31-3所示。

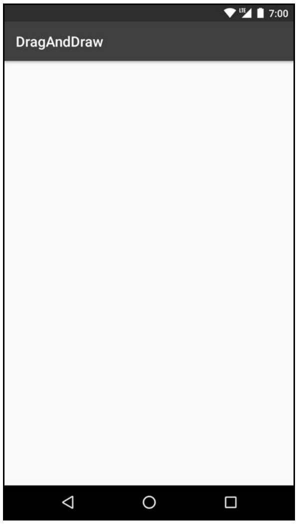


图31-3 未绘制的BoxDrawingView

接下来，让BoxDrawingView监听触摸事件，并实现在屏幕上绘制矩形框。

31.3 处理触摸事件

监听触摸事件的一种方式是使用以下View方法，设置一个触摸事件监听器：

```
public void setOnTouchListener(View.OnTouchListener l)
```

其工作方式与setOnClickListener(View.OnClickListener)相同。实现View.OnTouchListener接口，供触摸事件发生时调用。

不过，由于定制视图是View的子类，也可走捷径直接覆盖以下View方法：

```
public boolean onTouchEvent(MotionEvent event)
```

该方法接收一个MotionEvent类实例。MotionEvent类可用来描述包括位置和动作的触摸事件。动作用于描述事件所处的阶段。

动作常量	动作描述
ACTION_DOWN	手指触摸到屏幕
ACTION_MOVE	手指在屏幕上移动
ACTION_UP	手指离开屏幕
ACTION_CANCEL	父视图拦截了触摸事件

在onTouchEvent(...)实现方法中，可使用以下MotionEvent方法查看动作值：

```
public final int getAction()
```

在BoxDrawingView.java中，添加一个日志tag，然后实现onTouchEvent(...)方法记录可能发生的四个不同动作，如代码清单31-5所示。

代码清单31-5　实现BoxDrawingView视图类（BoxDrawingView.java）

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";
    ...
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        PointF current = new PointF(event.getX(), event.getY());
        String action = "";

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                action = "ACTION_DOWN";
                break;
            case MotionEvent.ACTION_MOVE:
                action = "ACTION_MOVE";
                break;
            case MotionEvent.ACTION_UP:
                action = "ACTION_UP";
                break;
            case MotionEvent.ACTION_CANCEL:
                action = "ACTION_CANCEL";
                break;
        }

        Log.i(TAG, action + " at x=" + current.x +
                ", y=" + current.y);

        return true;
    }
}
```

注意，X和Y坐标已经封装到PointF对象中。稍后，我们需要同时传递这两个坐标值。而Android提供的PointF容器类刚好满足了这一需求。

运行DragAndDraw应用并打开LogCat视图窗口。触摸屏幕并移动手指，查看BoxDrawingView接收的触摸动作的X和Y坐标记录。

跟踪运动事件

除了记录坐标，BoxDrawingView主要用于在屏幕上绘制矩形框。要实现这一目标，有几个问题需要解决。

首先，要知道定义矩形框的两个坐标点：原始坐标点（手指的初始位置），当前坐标点（手指的当前位置）。

其次，定义一个矩形框，还需追踪记录来自多个MotionEvent的数据。这些数据会保存在Box对象中。

新建一个Box类，用于表示一个矩形框的定义数据，如代码清单31-6所示。

代码清单31-6　添加Box类（Box.java）

```
public class Box {
    private PointF mOrigin;
    private PointF mCurrent;

    public Box(PointF origin) {
        mOrigin = origin;
        mCurrent = origin;
    }

    public PointF getCurrent() {
        return mCurrent;
    }

    public void setCurrent(PointF current) {
        mCurrent = current;
    }

    public PointF getOrigin() {
        return mOrigin;
    }
}
```

用户触摸BoxDrawingView视图界面时，新Box对象会创建并添加到现有矩形框数组中，如图31-4所示。

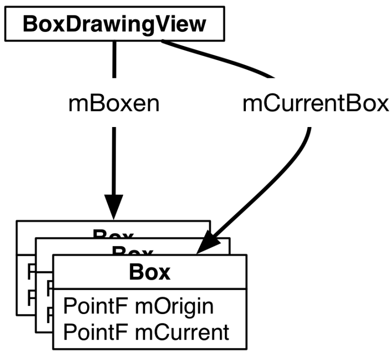


图31-4　DragAndDraw应用中的对象

回到BoxDrawingView类中，使用新Box对象跟踪绘制状态，如代码清单31-7所示。

代码清单31-7　添加拖曳生命周期方法（BoxDrawingView.java）

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";
```

```
    private Box mCurrentBox;
    private List<Box> mBoxen = new ArrayList<>();
    ...
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        PointF current = new PointF(event.getX(), event.getY());
        String action = "";

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                action = "ACTION_DOWN";
                // Reset drawing state
                mCurrentBox = new Box(current);
                mBoxen.add(mCurrentBox);
                break;
            case MotionEvent.ACTION_MOVE:
                action = "ACTION_MOVE";
                if (mCurrentBox != null) {
                    mCurrentBox.setCurrent(current);
                    invalidate();
                }
                break;
            case MotionEvent.ACTION_UP:
                action = "ACTION_UP";
                mCurrentBox = null;
                break;
            case MotionEvent.ACTION_CANCEL:
                action = "ACTION_CANCEL";
                mCurrentBox = null;
                break;
        }

        Log.i(TAG, action + " at x=" + current.x +
                ", y=" + current.y);

        return true;
    }
}
```

任何时候，只要接收到ACTION_DOWN动作事件，就以事件原始坐标新建Box对象并赋值给mCurrentBox，然后再添加到矩形框数组中。（31.4节处理定制绘制时，BoxDrawingView会在屏幕上绘制数组中的全部Box。）

用户手指在屏幕上移动时，mCurrentBox.mCurrent会得到更新。在取消触摸事件或用户手指离开屏幕时，清空mCurrentBox以结束屏幕绘制。已完成的Box会安全地存储在数组中，但它们再也不会受任何动作事件影响了。

注意ACTION_MOVE事件发生时调用的invalidate()方法。该方法会强制BoxDrawingView重新绘制自己。这样，用户在屏幕上拖曳时就能实时看到矩形框。这同时也引出了接下来的任务：在屏幕上绘出矩形框。

31.4 onDraw(...)方法内的图形绘制

应用启动后，所有视图都处于*无效状态*。也就是说，视图还没有绘制到屏幕上。为解决这个问题，Android调用了顶级View视图的draw()方法。这会引起自上而下的链式调用反应。首先，视图完成自我绘制，然后是子视图的自我绘制，再然后是子视图的子视图的自我绘制，如此调用下去直至继承结构的末端。当继承树中的所有视图都完成自我绘制后，最顶级View视图也就生效了。

为加入这种绘制，可覆盖以下View方法：

```
protected void onDraw(Canvas canvas)
```

前面，在onTouchEvent(MotionEvent)方法中响应ACTION_MOVE动作时，我们调用invalidate()方法再次让BoxDrawingView失效。这会迫使它重新自我绘制，并再次调用onDraw(Canvas)方法。

现在一起看看Canvas参数。Canvas和Paint是Android系统的两大绘制类。

- Canvas类拥有我们需要的所有绘制操作。其方法可决定绘在哪里以及绘什么，比如线条、圆形、字词、矩形等。
- Paint类决定如何绘制。其方法可指定绘制图形的特征，例如是否填充图形、使用什么字体绘制、线条是什么颜色等。

返回BoxDrawingView.java中，在BoxDrawingView的XML构造方法中创建两个Paint对象，如代码清单31-8所示。

代码清单31-8　创建Paint（BoxDrawingView.java）

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";

    private Box mCurrentBox;
    private List<Box> mBoxen = new ArrayList<>();
    private Paint mBoxPaint;
    private Paint mBackgroundPaint;
    ...
    // Used when inflating the view from XML
    public BoxDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);

        // Paint the boxes a nice semitransparent red (ARGB)
        mBoxPaint = new Paint();
        mBoxPaint.setColor(0x22ff0000);

        // Paint the background off-white
        mBackgroundPaint = new Paint();
        mBackgroundPaint.setColor(0xfff8efe0);
    }
}
```

有了Paint对象的支持，现在能在屏幕上绘制矩形框了，如代码清单31-9所示。

31

代码清单31-9　覆盖onDraw(Canvas)方法（BoxDrawingView.java）

```
    public BoxDrawingView(Context context, AttributeSet attrs) {
        ...
    }

    @Override
    protected void onDraw(Canvas canvas) {
        // Fill the background
        canvas.drawPaint(mBackgroundPaint);

        for (Box box : mBoxen) {
            float left = Math.min(box.getOrigin().x, box.getCurrent().x);
            float right = Math.max(box.getOrigin().x, box.getCurrent().x);
            float top = Math.min(box.getOrigin().y, box.getCurrent().y);
            float bottom = Math.max(box.getOrigin().y, box.getCurrent().y);

            canvas.drawRect(left, top, right, bottom, mBoxPaint);
        }
    }
```

以上代码的第一部分简单直接：使用米白背景paint，填充canvas以衬托矩形框。

然后，针对矩形框数组中的每一个矩形框，据其两点坐标，确定矩形框上下左右的位置。绘制时，左端和顶端的值作为最小值，右端和底端的值作为最大值。

完成位置坐标值计算后，调用`Canvas.drawRect(...)`方法，在屏幕上绘制红色矩形框。

运行DragAndDraw应用，尝试绘制一些红色矩形框，如图31-5所示。

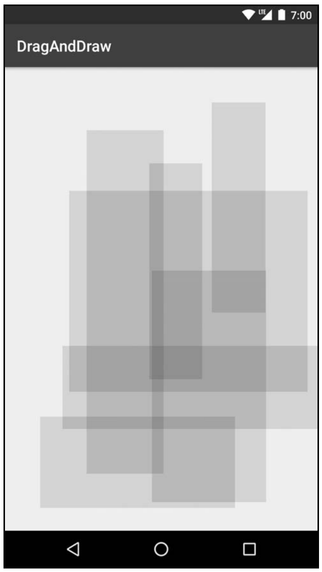


图31-5　程序员式的情绪表达

好了，一个捕捉其触摸事件并执行绘制的视图创建完成了。

31.5 挑战练习：设备旋转问题

设备旋转后，已绘制的矩形框会消失。要解决这个问题，可使用以下View方法：

```
protected Parcelable onSaveInstanceState()
protected void onRestoreInstanceState(Parcelable state)
```

以上方法的工作方式不同于Activity和Fragment的onSaveInstanceState(Bundle)方法。首先，View视图有ID时，才可以调用它们。其次，相较于Bundle参数，这些方法返回并处理的是实现Parcelable接口的对象。推荐使用Bundle，这样就不用自己实现Parcelable接口了。（Parcelable接口的实现很复杂，如有可能，应尽量避免。）

最后，还需要保存BoxDrawingView的View父视图的状态。在Bundle中保存super.onSaveInstanceState()方法结果，然后调用super.onRestoreInstanceState(Parcelable)方法把结果发送给超类。

31.6 挑战练习：旋转矩形框

请实现以两根手指旋转矩形框。这个练习有点难，想完成它，需在MotionEvent实现代码中处理多个触控点（pointer）。当然，还要旋转canvas。

要处理多点触摸，先清楚以下概念。

- ❑ pointer index：获知当前一组触控点中，动作事件对应的触控点。
- ❑ pointer ID：给予手势中特定手指一个唯一的ID。

pointer index可能会变，但pointer ID绝对不会变。

请查阅开发者文档，学习以下MotionEvent方法的使用：

```
public final int getActionMasked()
public final int getActionIndex()
public final int getPointerId(int pointerIndex)
public final float getX(int pointerIndex)
public final float getY(int pointerIndex)
```

另外，还需查查ACTION_POINTER_UP和ACTION_POINTER_DOWN常量的用法。

第 32 章

属性动画

32

写出没有崩溃性错误的代码，就能做出个勉强能用的应用。再多花点心思，或许就能做出用户想用、爱用的应用。这还不够，应用最好能模拟物理世界，给用户真实临场的感觉。

真实世界灵动多变。要让用户界面动起来，你需要让用户界面元素从一个位置动态移动到另一个位置。

本章，我们来开发一个模拟落日景象的应用。按住屏幕，太阳会慢慢落下海平面，天空的颜色随之不断变换，犹如真的日落。

32.1 建立场景

首先是创建动画场景。创建一个名为Sunset的新项目，确保minSdkVersion设置为API 19，并且使用空activity模板。将主activity命名为SunsetActivity。然后把之前项目的SingleFragmentActivity.java和activity_fragment.xml文件复制到当前项目。

海边落日光影变换，色彩斑斓。因此，需要准备一些色彩资源。在res/values目录中新建colors.xml文件。该资源文件内容定义如代码清单32-1所示。

代码清单32-1 添加落日色彩（res/values/colors.xml）

```
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="bright_sun">#fcfcb7</color>
    <color name="blue_sky">#1e7ac7</color>
    <color name="sunset_sky">#ec8100</color>
    <color name="night_sky">#05192e</color>
    <color name="sea">#224869</color>
</resources>
```

虽然矩形视图模拟天空和大海的效果还可以，但没人见过方方长长的太阳吧？技术实现简单可不是理由。所以，在res/drawable/目录，新建一个sun.xml椭圆形drawable资源，如代码清单32-2所示。

代码清单32-2 添加模拟太阳的XML drawable（res/drawable/sun.xml）

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <solid android:color="@color/bright_sun" />
</shape>
```

在矩形视图上显示这个椭圆形drawable，就会看到一个圆。现在，用户一定会点头赞许，仿佛看到真正的太阳挂在天空。

接下来，使用一个完整的布局文件构建整个场景。该布局会在稍后创建的SunsetFragment中使用，因此直接命名为fragment_sunset.xml。布局定义如代码清单32-3所示。

代码清单32-3 创建落日场景布局（res/layout/fragment_sunset.xml）

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <FrameLayout
        android:id="@+id/sky"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.61"
        android:background="@color/blue_sky">
        <ImageView
            android:id="@+id/sun"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_gravity="center"
            android:src="@drawable/sun" />
    </FrameLayout>

    <View
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.39"
        android:background="@color/sea" />
</LinearLayout>
```

现在预览布局。怎么样，大海蔚蓝，天蓝日落，多么动人的画面啊！这不禁让人念起那去往海滩或坐船出海的过往旅程。

32

布局搞定了，下面要编写代码让应用跑起来。创建一个SunsetFragment类，并在其中新增一个newInstance()方法，然后，在onCreateView(...)方法中，实例化fragment_sunset布局并返回结果视图，如代码清单32-4所示

代码清单32-4 创建SunsetFragment类（SunsetFragment.java）

```
public class SunsetFragment extends Fragment {

    public static SunsetFragment newInstance() {
        return new SunsetFragment();
    }
```

```java
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_sunset, container, false);

        return view;
    }
}
```

为显示SunsetFragment，让SunsetActivity类继承SingleFragmentActivity类，如代码清单32-5所示。

代码清单32-5 创建SunsetFragment类（SunsetActivity.java）

```java
public class SunsetActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return SunsetFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

}
```

运行Sunset应用。一切正常的话，可看到如图32-1所示的画面。

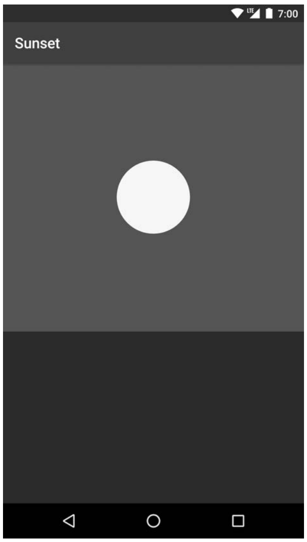


图32-1 落日徐徐

32.2　简单属性动画

创建完落日场景，现在要实现太阳徐徐落下海平面的动画效果。

首先，你需要在fragment中获取一些必要的信息。在onCreateView(...)方法中，获取要控制的视图并存入相应变量中备用，如代码清单32-6所示。

代码清单32-6　获取视图引用（SunsetFragment.java）

```
public class SunsetFragment extends Fragment {

    private View mSceneView;
    private View mSunView;
    private View mSkyView;

    public static SunsetFragment newInstance() {
        return new SunsetFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_sunset, container, false);

        mSceneView = view;
        mSunView = view.findViewById(R.id.sun);
        mSkyView = view.findViewById(R.id.sky);

        return view;
    }
}
```

做完了准备工作，接下来就是编码实现了。从技术上讲，所谓太阳落下海平面，实际就是平滑地移动mSunView视图，直到它的顶部刚好与海平面的顶部边缘重合。这可以通过改变mSunView视图顶部的坐标位置来实现。

显然，这需要知道动画的开始和结束点。这个任务就交给startAnimation()方法处理，如代码清单32-7所示。

32

代码清单32-7　获取视图的顶部坐标位置（SunsetFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    ...
}

private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();
}
```

和View视图类的getBottom()、getRight()和getLeft()方法一样，getTop()方法可以返回自己的local layout rect。视图的local layout rect是其相对父视图的位置和其尺寸大小的描述。视图一旦实例化，这些值都是相对固定的。虽然可以修改这些值，从而改变视图的位置，但不推荐这么做。要知道，每次布局切换时，这些值都会被重置，所以才会有相对固定一说。

无论怎样，动画的开始点都是mSunView视图的顶部位置。结束点是其父视图mSkyView的底部位置。移动距离是调用getHeight()方法返回的mSkyView高度。除了getHeight()方法，调用getBottom()和getTop()方法并取差值也能获得相同结果。

知道了动画的开始和结束点，创建一个ObjectAnimator对象执行动画，如代码清单32-8所示。

代码清单32-8　创建模拟太阳的animator对象（SunsetFragment.java）

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
            .ofFloat(mSunView, "y", sunYStart, sunYEnd)
            .setDuration(3200);

    heightAnimator.start();
}
```

在onCreateView()方法中，为mSceneView视图设置监听器。只要用户点击它，就调用startAnimation()方法执行动画，如代码清单32-9所示。

代码清单32-9　响应触摸，执行动画（SunsetFragment.java）

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_sunset, container, false);

    mSceneView = view;
    mSunView = view.findViewById(R.id.sun);
    mSkyView = view.findViewById(R.id.sky);

    mSceneView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startAnimation();
        }
    });

    return view;
}
```

运行Sunset应用。点击应用界面任意处，欣赏一段美丽的落日动画，如图32-2所示。

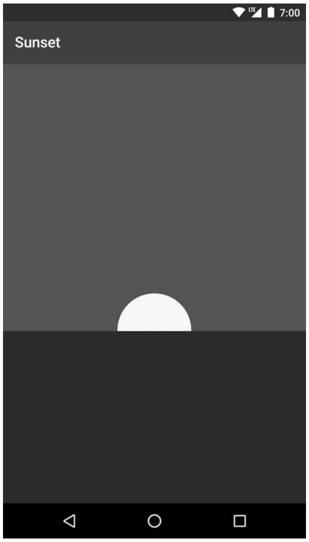


图32-2　落日

最后，来看看这段动画的实现原理：ObjectAnimator是个属性动画制作对象。要获得某种动画效果，传统方式是设法在屏幕上移动视图，而属性动画制作对象却另辟蹊径：以一组不同的参数值反复调用属性设置方法。

调用以下方法可以创建ObjectAnimator对象：

```
ObjectAnimator.ofFloat(mSunView, "y", 0, 1)
```

上例中，新建ObjectAnimator一旦启动，就会以从0开始递增的参数值反复调用mSunView.setY(float)方法：

```
mSunView.setY(0);
mSunView.setY(0.02);
mSunView.setY(0.04);
mSunView.setY(0.06);
mSunView.setY(0.08);
...
```

32

直到调用mSunView.setY(1)为止。这个0~1区间参数值的确定过程又称为interpolation。可以想象到，在这个interpolation过程中，即便很短暂，确定相邻参数值也是要耗费时间的；由于人眼的视觉暂留现象，动画效果就形成了。

32.2.1　视图属性转换

想让视图动起来的话，仅仅靠属性动画制作对象是不切实际的，尽管它确实很有用。因此，有了属性转换（transformation properties）这个合作伙伴。

前面说过，视图都有local layout rect（视图实例化时被赋予的位置及大小尺寸参数值）。知道了视图属性值（local layout rect），就可以改变这些属性值，从而实现四处移动视图。这种做法就

叫作属性转换。例如，利用rotation、pivotX和pivotY这三个参数可以旋转视图（见图32-3）；利用scaleX和scaleY可以缩放视图（见图32-4）；而利用translationX和translationY可以四处移动视图（见图32-5）。

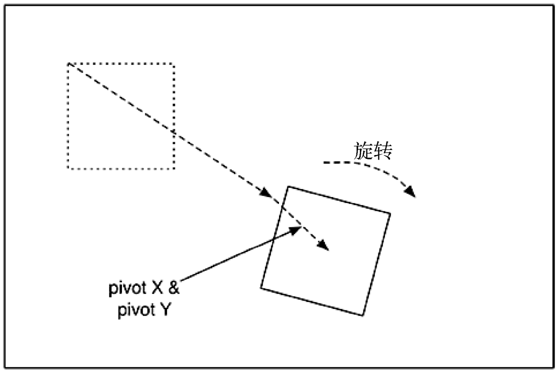


图32-3　视图旋转

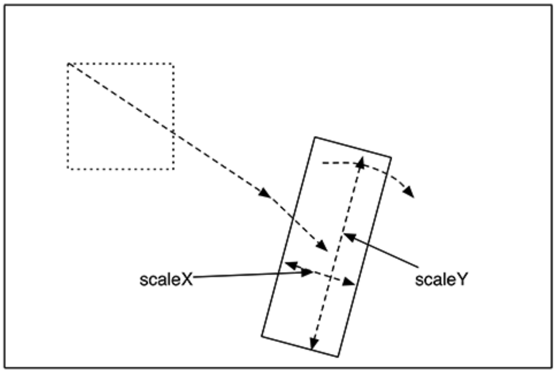


图32-4　视图缩放

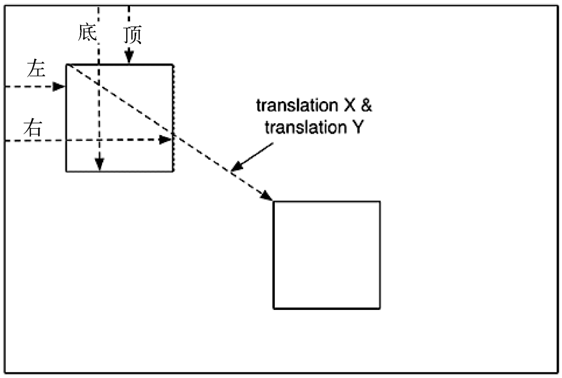


图32-5　视图移动

视图的所有这些属性值都有getter方法和setter方法。例如，调用`getTranslationX()`方法就能得到`translationX`值，调用`setTranslationX(float)`方法就能设置`translationX`值。

那么y属性有什么作用呢？实际上，x和y属性是以布局坐标为参考值设立的一种便利开发的属性值。例如，简单写几行代码，就可以把视图置于某个X和Y坐标确定的位置。分析其背后原理可知，这就是通过修改`translationX`和`translationY`属性值来实现的。所以，调用`mSunView.setY(50)`方法就等同于：

```
mSunView.setTranslationY(50 - mSunView.getTop())
```

32.2.2 使用不同的 interpolator

目前，Sunset应用的动画效果还不够完美。假设太阳一开始静止于天空，在进入落下的动画时，应该有个加速过程。这也好办，使用`TimeInterpolator`就可以了。这个`TimeInterpolator`的作用就是：改变*A*点到*B*点的动画效果。

在`startAnimation()`方法中，添加代码清单32-10所示代码，使用一个`AccelerateInterpolator`对象实现太阳加速落下的特效。

代码清单32-10 添加加速特效（SunsetFragment.java）

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
            .ofFloat(mSunView, "y", sunYStart, sunYEnd)
            .setDuration(3000);
    heightAnimator.setInterpolator(new AccelerateInterpolator());

    heightAnimator.start();
}
```

重新运行Sunset应用。点击屏幕观察动画效果。这次，太阳先是慢慢落下，然后朝着海平面方向加速坠落。

使用不同的`TimeInterpolator`对象可实现不同的动画特效。想要了解Android自带的`TimeInterpolator`还有哪些，请参阅`TimeInterpolator`参考文档的Known Indirect Subclasses部分。

32

32.2.3 色彩渐变

优化完落日的动画效果，接着处理天空随日落所呈现的色彩变换效果。在`onCreateView(...)`方法中，获取colors.xml文件定义的色彩资源并存入相应的实例变量，如代码清单32-11所示。

代码清单32-11 取出日落色彩资源（SunsetFragment.java）

```
public class SunsetFragment extends Fragment {
    ...
```

```
    private View mSkyView;

    private int mBlueSkyColor;
    private int mSunsetSkyColor;
    private int mNightSkyColor;
    ...
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        mSkyView = view.findViewById(R.id.sky);

        Resources resources = getResources();
        mBlueSkyColor = resources.getColor(R.color.blue_sky);
        mSunsetSkyColor = resources.getColor(R.color.sunset_sky);
        mNightSkyColor = resources.getColor(R.color.night_sky);

        mSceneView.setOnClickListener(new View.OnClickListener() {
            ...
        });

        return view;
    }
```

在startAnimation()方法中，参照代码清单32-12再添加一个ObjectAnimator，实现天空色彩从mBlueSkyColor到mSunsetSkyColor变换的动画效果。

代码清单32-12　实现天空的色彩变换（SunsetFragment.java）

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
            .ofFloat(mSunView, "y", sunYStart, sunYEnd)
            .setDuration(3000);
    heightAnimator.setInterpolator(new AccelerateInterpolator());

    ObjectAnimator sunsetSkyAnimator = ObjectAnimator
            .ofInt(mSkyView, "backgroundColor", mBlueSkyColor, mSunsetSkyColor)
            .setDuration(3000);

    heightAnimator.start();
    sunsetSkyAnimator.start();
}
```

天空的动画效果似乎就这么完成了。运行SunSet应用看看吧。怎么，似乎不大对劲啊！从蓝色到橘黄色，天空的色彩变化太夸张了，一点都不自然。

仔细分析就知道，颜色int数值并不是个简单的数字。它实际是由四个较小数字转换而来。因此，只有知道颜色的组成奥秘，ObjectAnimator对象才能合理地确定蓝色和橘黄色之间的中间值。

不过，知道如何确定颜色中间值还不够，ObjectAnimator还需要一个TypeEvaluator子类

的协助。TypeEvaluator能帮助ObjectAnimator对象精确地计算开始到结束间的递增值。Android提供的这个TypeEvaluator子类叫作ArgbEvaluator，如代码清单32-13所示。

代码清单32-13 使用ArgbEvaluator（SunsetFragment.java）

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
            .ofFloat(mSunView, "y", sunYStart, sunYEnd)
            .setDuration(3000);
    heightAnimator.setInterpolator(new AccelerateInterpolator());

    ObjectAnimator sunsetSkyAnimator = ObjectAnimator
            .ofInt(mSkyView, "backgroundColor", mBlueSkyColor, mSunsetSkyColor)
            .setDuration(3000);
    sunsetSkyAnimator.setEvaluator(new ArgbEvaluator());

    heightAnimator.start();
    sunsetSkyAnimator.start();
}
```

再次运行Sunset应用。夕阳西下，从蓝色到橘黄色，色彩的过渡终于自然了。

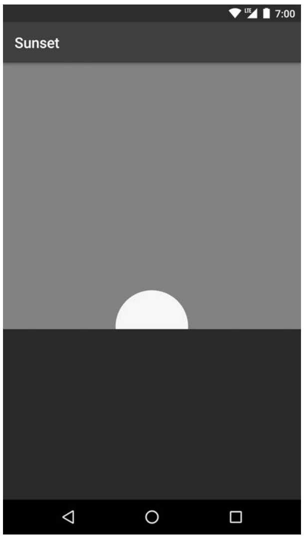


图32-6 天空的色彩随日落变换

32.3 播放多个动画

有时，你需要同时执行一些动画。这很简单，同时调用start()方法就行了。

但是，假如要像编排舞步那样编排多个动画的执行，事情就没那么简单了。例如，为实现完整的日落景象，太阳落下去之后，天空应该从橘黄色再转为午夜蓝。

办法总是有的，你可以使用AnimatorListener。AnimatorListener会让你知道动画什么时候结束。这样，执行完第一个动画，就可以接力执行第二个夜空变化的动画。然而，理论分析很简单，实际去做的话，少不了要准备多个监听器，这也很麻烦。好在Android还设计了方便又简单的AnimatorSet。下面就来学习使用它。

首先，删除掉原来的动画启动代码，并添加夜空变化的动画代码，如代码清单32-14所示

代码清单32-14　创建夜空动画（SunsetFragment.java）

```
private void startAnimation() {
    ...
    sunsetSkyAnimator.setEvaluator(new ArgbEvaluator());

    ObjectAnimator nightSkyAnimator = ObjectAnimator
            .ofInt(mSkyView, "backgroundColor", mSunsetSkyColor, mNightSkyColor)
            .setDuration(1500);
    nightSkyAnimator.setEvaluator(new ArgbEvaluator());

    heightAnimator.start();
    sunsetSkyAnimator.start();

}
```

然后，创建并执行一个动画集，如代码清单32-15所示。

代码清单32-15　创建动画集（SunsetFragment.java）

```
private void startAnimation() {
    ...
    ObjectAnimator nightSkyAnimator = ObjectAnimator
            .ofInt(mSkyView, "backgroundColor", mSunsetSkyColor, mNightSkyColor)
            .setDuration(1500);
    nightSkyAnimator.setEvaluator(new ArgbEvaluator());

    AnimatorSet animatorSet = new AnimatorSet();
    animatorSet
            .play(heightAnimator)
            .with(sunsetSkyAnimator)
            .before(nightSkyAnimator);
    animatorSet.start();
}
```

说白了，AnimatorSet就是可以放在一起执行的动画集。可以用好几种方式创建动画集，但使用上述代码中的play(Animator)方法最容易。

调用play(Animator)方法之前，要先创建一个AnimatorSet.Builder对象，然后利用它创建链式方法调用。传入play(Animator)方法的Animator是链首。所以，以上代码中的链式调用就可以这样解读：协同执行heightAnimator和sunsetSkyAnimator动画，在nightSkyAnimator之前执行heightAnimator动画。在实际开发中，可能会用到更复杂的动画集。这也没问题，需

要的话，可以多次调用`play(Animator)`方法。

再次运行Sunset应用。用心感受下这幅动人祥和的画面，真是太棒了！

32.4 深入学习：其他动画 API

除了广受欢迎的属性动画，Android动画工具箱里还有一些其他动画工具。不管用不用，花点时间了解一下总没错。

32.4.1 传统动画工具

Android有个叫作`android.view.animation`的动画工具类包。Honeycomb发布时，又引入了一个更新的`android.animation`包。这是两个不同的包，请注意区分。

它们都是传统的动画工具包，简单了解就可以了。注意到了吗？本章使用的动画工具类的类名都为animaTOR。如果遇到animaTION这样的类名，就能断定它来自传统动画工具包，直接忽略好了！

32.4.2 转场

Android 4.4引入了新的视图转场框架。从一个activity小视图动态弹出另一个放大版activity视图就可以使用转场框架实现。

实际上，转场框架的工作原理很简单：定义一些场景，它们代表各个时点的视图状态，然后按照一定的逻辑切换场景。场景在XML布局文件中定义，转场在XML动画文件中定义。

在本章日落例子中，activity已经运行了，这种情况就不太适合使用转场框架，所以我们用了强大的属性动画框架。

再以CriminalIntent应用中处理crime图片为例，如果想实现以弹窗展示放大版图片这样的动画效果，首先要知道照片放在哪里，其次是如何在对话框里布置新图片。显然，对于这类布局动态转场任务，转场框架比`ObjectAnimator`更能胜任。

32.5 挑战练习

首先，让日落可逆。也就是说，点击屏幕，等太阳落下后，再次点击屏幕，让太阳升起来。动画集不能逆向执行，因此，你需要新建一个`AnimatorSet`。

第二个挑战是添加太阳动画特效，让它有规律地放大、缩小或是加一圈旋转的光线。（这实际是反复执行一段动画特效，可考虑使用`ObjectAnimator`的`setRepeatCount(int)`方法。）

另外，海面上要是有太阳的倒影就更真实了。

最后，再来看个颇具挑战的练习。在日落过程中实现动画反转。在太阳慢慢下落时点击屏幕，让太阳无缝回升至原来所在的位置。或者，在太阳落下进入夜晚时点击屏幕，让太阳重新升回天空，就像日出。

第 33 章

地理位置和Play服务

本章，我们来编写一个叫作Locatr的应用。Locatr支持基于地理位置的Flickr图片搜索。它首先会定位用户的当前位置，然后搜索附近的图片（如图33-1所示）。下一章，应用升级后，图片还会显示在地图上。

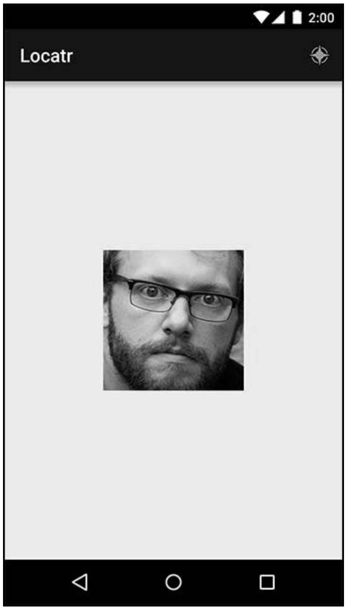


图33-1　Locatr应用效果图

用户当前位置定位简单又有趣。但需要把一个叫作Google Play Service的非标准库整合进应用。

33.1　地理位置和定位类库

为啥要整合其他库呢？稍后会解答。现在我们先看看原生Android设备能提供啥定位数据。再看看Android又提供了哪些工具，使用它们又能得到什么样的定位数据。

Android原生系统提供了一些基本地理位置API。使用这些API，应用可以从各种信息源接收地理位置数据。对大多数手机来说，这些信息源是指来自GPS定位仪（数据较精准）以及来自手机基站或Wi-Fi连接的地理位置数据。这些API在刚有Android的时候就有了，可以在`android.location`库包中找到。

虽然它们早已存在，但无法满足所有需求场景。真实世界里的应用往往有这样的需求，“请尽可能给我最精确的定位数据，费不费电没关系”，或者“我需要定位数据，最好是不要太费电”。而像“请启动GPS定位仪，给个定位数据就行了”这样没啥要求的却不多见。

一旦你带着手机到处跑，问题更加凸显。如果在户外，GPS最合适。如果GPS没信号，手机基站定位也不错。如果两者都不行，凑合使用加速感应器和陀螺仪也总比完全没方向要强。

过去，为获得定位数据，严肃应用必须手动调用上述各种数据源，并且要根据不同场景适时切换。这实施起来比较复杂，难以把握。

Google Play Service

考虑到上述情况，就迫切需要更好的定位API。然而，如果要把这样的API加入标准库，开发人员不知道猴年马月才能用得到。这就比较尴尬了，操作系统可是拥有定位所需的一切硬件的：GPS、手机基站定位等。

还好，标准库并不是Google发布API的唯一途径。除了标准库，它还提供了Google Play Service。这是一套随Google Play商店应用安装的常用服务。为解决定位疑难问题，Google在Play服务中提供了叫作Fused Location Provider的全新定位服务。

既然这些库包含在其他应用里，那么首先要安装这些应用。同时，这也意味着要安装Play商店应用并且随时保持更新。而且，你的应用也要通过Play商店发布。如果做不到，不好意思，请使用其他定位API吧。

对于Locatr应用，如果使用物理设备，请确保Play商店应用已更新到最新。要是使用模拟器呢？别担心，稍后就会提供解决办法。

33.2 创建 Locatr 项目

理论知识已足够，可以创建应用了。在Android Studio中，创建一个叫作Locatr的新项目。创建一个空activity并将其命名为`LocatrActivity`，最低SDK版本为API 19。然后将SingleFragment-Activity.java和activity_fragment.xml文件分别复制到Locatr项目的对应目录。

Locatr应用也需要搜索Flickr，所以为了方便，它需要PhotoGallery应用中的一些查询代码。从PhotoGallery的随书项目文件中找到FlickrFetchr.java和GalleryItem.java（建议使用第27章的文件），将它们复制到Locatr项目的对应目录。

Locatr应用的用户界面稍后创建。如果使用模拟器，请继续阅读下一节配置测试环境。否则，请直接跳到33.4节。

33.3　Play 服务定位和模拟器

如果使用AVD模拟器，请确保模拟器镜像已更新到最新版本。

为更新镜像文件，打开SDK管理器（Tools → Android → SDK Manager）。找到你要使用的Androd版本，并确认Google APIs System Images已安装并更新至最新。如提示有更新，请按照提示完成更新（如图33-2所示）。

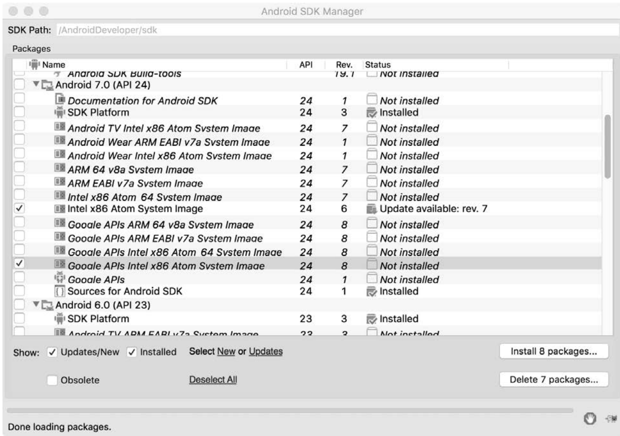


图33-2　确保模拟器已更新

AVD模拟器也应有个支持指定版本Google API的目标操作系统。创建模拟器时，选择目标操作系统版本（with Google APIs）时会看到API级别。对于Locatr项目，选择API 21级或更高即可（如图33-3所示）。

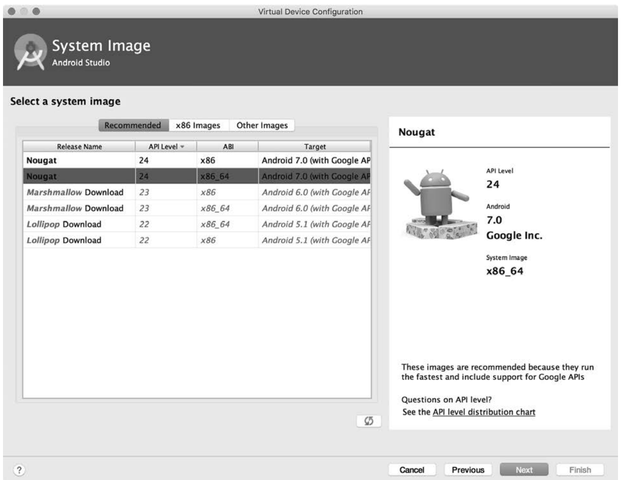


图33-3　选择API级别

如果先前已搭建了合适的模拟器，同样需要更新至最新版本。完成更新后，记得重启模拟器让其生效。

模拟定位数据

在模拟器上，你也需要不断更新的地理位置信息用于测试。Android Studio提供模拟器控制面板，可以发送地理位置坐标给模拟器；但它只适用于以前的定位服务，不能用于Fused Location Provider。看来，只能以代码的方式发送地理位置信息了。

在Big Nerd Ranch培训基地，只要你感兴趣，任何主题我们都愿意细细讲给你听。不过，在遭遇了狩猎教学大失败之后，我们终于明白不能这样教了。现在，我们更实际，更乐意传授一些学生最急需、更有用的知识。所以，我们放弃了教授如何编写模拟数据发送代码，而是为你写了一个叫作MockWalker的独立应用。要使用它，请从以下地址下载并安装其APK：https://www.bignerdranch.com/solutions/MockWalker.apk。

最容易安装的方式是打开模拟器上的浏览器应用，输入地址，如图33-4所示。

下载完成后，点击工具栏的下载通知项打开这个APK完成安装，如图33-5所示。

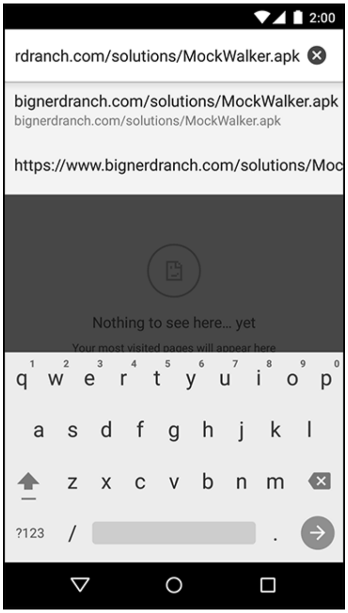


图33-4　输入下载地址

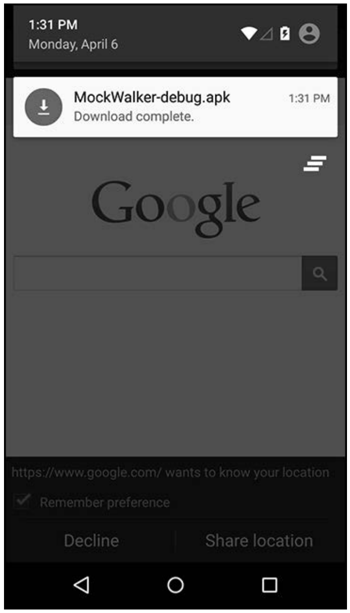


图33-5　打开下载文件

MockWalker会触发一个模拟步行行为，通过服务把模拟地理位置数据发送给Fused Location Provider。应用运行后，它会假装在亚特兰大的柯克伍德附近转悠。

服务运行时，只要Locatr应用向Fused Location Provider请求定位数据，它就会收到MockWalker发布的地理位置数据（如图33-6所示）。

33



图33-6　运行MockWalker

运行MockWalker应用并点击Start按钮。即使退出应用，服务也会一直在后台运行。（千万不要退出模拟器。）使用完毕或不再需要的话，就重新打开MockWalker应用，点击Stop按钮关闭。

如果想了解MockWalker应用的工作原理，可以查看它的源码（见随书代码文件）。为管理持续的地理位置更新，该应用使用了RxJava和sticky前台服务技术。

33.4　创建 Locatr 应用

接下来，为Locatr应用创建用户界面。首先，为搜索按钮添加字符串资源，如代码清单33-1所示。

代码清单33-1　添加搜索按钮文字（res/values/strings.xml）

```
<resources>
    <string name="app_name">Locatr</string>

    <string name="search">Find an image near you</string>
</resources>
```

老规矩，你需要一个fragment，所以重命名activity_locatr.xml布局文件为fragment_locatr.xml。修改`RelativeLayout`控件，使用一个`ImageView`控件显示搜索到的图片，如图33-7所示。（padding属性值不重要，如果模板自带，可直接删除。）

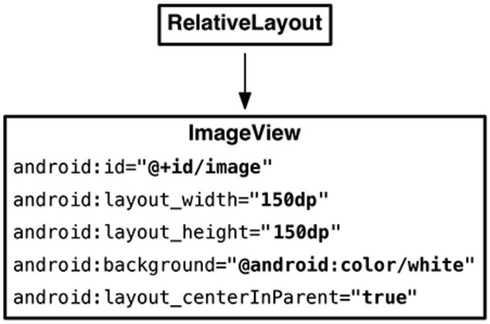


图33-7　Locatr应用的布局（res/layout/fragment_locatr.xml）

Locatr应用还需要一个按钮触发搜索。这里，可以利用工具栏来实现。创建res/menu/fragment_locatr.xml菜单文件。然后添加菜单项以显示一个搜索按钮，如代码清单33-2所示。（没错，菜单文件名和res/layout/fragment_locatr.xml一样。这没问题，菜单资源和布局资源的命名空间是不同的。）

代码清单33-2　配置菜单（res/menu/fragment_locatr.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_locate"
        android:icon="@android:drawable/ic_menu_compass"
        android:title="@string/search"
        android:enabled="false"
        app:showAsAction="ifRoom"/>
</menu>
```

按钮默认是禁用的。后面，连上Play服务之后就启用它。

现在，创建一个名为`LocatrFragment`的子类，继承`Fragment`，并在其中实例化布局，显示`ImageView`视图，如代码清单33-3所示。

代码清单33-3　创建`LocatrFragment`（LocatrFragment.java）

```
public class LocatrFragment extends Fragment {
    private ImageView mImageView;

    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_locatr, container, false);

        mImageView = (ImageView) v.findViewById(R.id.image);

        return v;
```

```
        }
    }
```

接着，完成菜单项的创建，如代码清单33-4所示。

代码清单33-4　添加菜单（LocatrFragment.java）

```
public class LocatrFragment extends Fragment {
    private ImageView mImageView;

    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.fragment_locatr, menu);
    }
}
```

最后，把`LocatrFragment`交给`LocatrActivity`托管，如代码清单33-5所示。

代码清单33-5　托管Locatr fragment（LocatrActivity.java）

```
public class LocatrActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return LocatrFragment.newInstance();
    }
}
```

至此，Locatr应用的用户界面搞定了。接下来是配置Play服务。

33.5　配置 Google Play 服务

要使用Fused Location Provider获取地理位置，就需要使用Google Play服务。要让这些服务运行起来，还需要一些基础准备工作。

首先是添加Google Play服务库依赖。服务本身是在Play应用里运行的，但Google Play服务库包含所有和它们交互的代码。

打开app模块设置（File → Project Structure）。在app模块的dependencies选项页，点击+按钮添加库依赖。添加的时候，要完整输入com.google.android.gms:play-services-location:10.0.1依赖库名。这是Play服务的定位类库。

随着代码的迭代，这个库的版本号一直在变。如果想知道最新的版本号，可使用play-services关键字搜索依赖库。如果想使用最新版本，可使用具体的版本号指定要使用的依赖库（play-services-location:x.x.x）。

如果有多个版本，到底该用哪个呢？实践表明，尽量使用最新的。不过，我们无法保证本章代码一定能与未来版本的依赖库兼容，所以，学习本章时，请使用10.0.1版本。

既然依赖的服务在设备上的其他应用中运行，就不敢保证Play服务库一定可用。接下来，需要在代码中检测是否有可用的Play服务。更新主activity执行必要的检查，如代码清单33-6所示。

代码清单33-6　检测Play服务（LocatrActivity.java）

```
public class LocatrActivity extends SingleFragmentActivity {
    private static final int REQUEST_ERROR = 0;

    @Override
    protected Fragment createFragment() {
        return LocatrFragment.newInstance();
    }

    @Override
    protected void onResume() {
        super.onResume();

        GoogleApiAvailability apiAvailability = GoogleApiAvailability.getInstance();
        int errorCode = apiAvailability.isGooglePlayServicesAvailable(this);

        if (errorCode != ConnectionResult.SUCCESS) {
            Dialog errorDialog = apiAvailability
                    .getErrorDialog(errorCode, this, REQUEST_ERROR,
                            new DialogInterface.OnCancelListener() {

                                @Override
                                public void onCancel(DialogInterface dialog) {
                                    // Leave if services are unavailable.
                                    finish();
                                }
                            });

            errorDialog.show();
        }
    }
}
```

通常，我们不会像这样使用`Dialog`。不过，这里`LocatrActivity`每次启动后的`errorCode`值都一样，所以`Dialog`都能正确显示。

地理位置定位权限

Locatr应用还需要地理位置定位权限才能工作。定位相关的权限有两个：android.permission.ACCESS_FINE_LOCATION和android.permission.ACCESS_COARSE_LOCATION。精准定位来自GPS定位仪；非精准定位来自手机基站或Wi-Fi接入点。

Locatr应用需要精准定位数据，所以肯定要添加ACCESS_FINE_LOCATION权限。但最好也一并把ACCESS_COARSE_LOCATION加上，因为万一GPS没信号，至少还可以尝试使用手机基站或Wi-Fi定位嘛。

Locatr应用需要搜索Flickr，因此，添加完定位权限，顺手再把网络使用权限添加上，如代码清单33-7所示。

代码清单33-7　添加权限（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.locatr" >

    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
        android:name="android.permission.INTERNET" />
    ...
</manifest>
```

不像INTERNET权限，FINE_LOCATION和COARSE_LOCATION都属于危险型权限。光在manifest中配置还够安全，你还需要运行时请求去用它们。相关权限使用代码稍后会写。现在，我们继续配置Google Play服务。

33.6　使用 Google Play 服务

要使用Play服务，还需要创建一个客户端类。这个客户端是个GoogleApiClient类实例。在Play服务参考文档区（http://developer.android.com/reference/gms-packages.html），可以找到这个类（和其他所有Play服务类）的使用说明。

为创建客户端，先创建一个GoogleApiClient.Builder。然后，使用我们要使用的API配置它。最后，调用build()方法创建实例。

在onCreate(Bundle)方法中，创建一个GoogleApiClient.Builder实例，然后，把定位服务API添加给它。如代码清单33-8所示。

代码清单33-8　创建GoogleApiClient（LocatrFragment.java）

```
public class LocatrFragment extends Fragment {
    private ImageView mImageView;
    private GoogleApiClient mClient;
```

```
    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);

        mClient = new GoogleApiClient.Builder(getActivity())
                .addApi(LocationServices.API)
                .build();
    }
```

创建了mClient客户端之后，需要连接它。Google推荐在onStart()方法里连接客户端，在onStop()方法里断开连接。调用客户端的connect()方法会改变菜单按钮的行为，所以要调用invalidateOptionsMenu()方法刷新它的状态（稍后还会再次调用它：被告知已连接的时候。)，如代码清单33-9所示。

代码清单33-9　连接和断开连接（LocatrFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
}

@Override
public void onStart() {
    super.onStart();

    getActivity().invalidateOptionsMenu();
    mClient.connect();
}

@Override
public void onStop() {
    super.onStop();

    mClient.disconnect();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
```


如果连不上客户端，应用就什么也做不了。所以，为反映客户端连接状况，按钮的启用和禁用状态应作相应切换，如代码清单33-10所示。

代码清单33-10　更新菜单按钮状态（LocatrFragment.java）

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
```

```
        inflater.inflate(R.menu.fragment_locatr, menu);

        MenuItem searchItem = menu.findItem(R.id.action_locate);
        searchItem.setEnabled(mClient.isConnected());
    }
```

发现连上客户端后，再添加另一个getActivity().invalidateOptionsMenu()调用更新菜单项。连接状态信息要通过两个回调接口传递：ConnectionCallbacks和OnConnectionFailed-Listener。在onCreate(Bundle)中，实现一个ConnectionCallbacks监听器，如代码清单33-11所示。

代码清单33-11　监听连接事件（LocatrFragment.java）

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);

        mClient = new GoogleApiClient.Builder(getActivity())
                .addApi(LocationServices.API)
                .addConnectionCallbacks(new GoogleApiClient.ConnectionCallbacks() {
                    @Override
                    public void onConnected(Bundle bundle) {
                        getActivity().invalidateOptionsMenu();
                    }

                    @Override
                    public void onConnectionSuspended(int i) {

                    }
                })
                .build();
    }
```

好奇的话，还可以实现一个OnConnectionFailedListener监听器，看看会有什么结果。当然，这不是必须的。

现在，Google Play服务可用了。

33.7　基于地理位置的 Flickr 搜索

接下来是实现基于地理位置的Flickr搜索。要完成这个任务，除了常规搜索功能外，还要附加经纬度地理位置信息。

Android定位API是在Location中封装这些地理位置信息的。所以，新写一个buildUrl(...)覆盖方法，从传入的Location对象中取出地理位置信息，用以创建我们需要的搜索Url，如代码清单33-12所示。

代码清单33-12　创建buildUrl(Location)方法（FlickrFetchr.java）

```
    private String buildUrl(String method, String query) {
```

```
    ...
}

private String buildUrl(Location location) {
    return ENDPOINT.buildUpon()
            .appendQueryParameter("method", SEARCH_METHOD)
            .appendQueryParameter("lat", "" + location.getLatitude())
            .appendQueryParameter("lon", "" + location.getLongitude())
            .build().toString();
}
```

然后，再写一个匹配的searchPhotos(Location)方法，如代码清单33-13所示。

代码清单33-13 创建searchPhotos(Location)方法（FlickrFetchr.java）

```
public List<GalleryItem> searchPhotos(String query) {
    ...
}

public List<GalleryItem> searchPhotos(Location location) {
    String url = buildUrl(location);
    return downloadGalleryItems(url);
}
```

33.8 获取定位数据

现在，万事俱备，只缺地理位置信息了。有一个类名叫FusedLocationProviderApi，顾名思义，要使用Fused Location Provider API就全靠它了。这个类有一个名为FusedLocationApi的实例，它是LocationServices对象中的一个单例对象。

要通过这个API获取地理位置信息，首先要使用LocationRequest对象创建一个定位请求。编写一个findImage()方法创建并配置我们需要的定位请求，如代码清单33-14所示。（有两个LocationRequest类可用,这里要用com.google.android.gms.location.LocationRequest类。）

代码清单33-14 创建定位请求（LocatrFragment.java）

```
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        ...
    }

    private void findImage() {
        LocationRequest request = LocationRequest.create();
        request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        request.setNumUpdates(1);
        request.setInterval(0);
    }
}
```

有好几个参数可以配置LocationRequest定位请求对象。

33

- 时间间隔（interval）：地理位置更新的频繁度。
- 更新次数（number of updates）：地理位置应该更新多少次。
- 优先级（priority）：是省电优先，还是定位精准度优先。
- 失效（expiration）：定位请求是否会失效。如果会，什么时候失效。
- 最小位移（smallest displacement）：触发位置更新，设备需移动的最小距离（以米单位）。

默认创建的LocationRequest定位请求的精度是街区级的，更新也很慢。所以，通过修改优先级、更新次数和时间间隔，我们可创建既精准又快速的定位请求。为获取不间断实时数据，我们把时间间隔（interval）设置为0。

接下来就是发出定位请求并从Location那里监听反馈。这需要添加一个LocationListener。有两个版本的LocationListener可用，这里选择的是com.google.android.gms.location.LocationListener，如代码清单33-15所示。

代码清单33-15　发送定位请求（LocatrFragment.java）

```
public class LocatrFragment extends Fragment {
    private static final String TAG = "LocatrFragment";
    ...
    private void findImage() {
        LocationRequest request = LocationRequest.create();
        request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        request.setNumUpdates(1);
        request.setInterval(0);
        LocationServices.FusedLocationApi
                .requestLocationUpdates(mClient, request, new LocationListener() {
                    @Override
                    public void onLocationChanged(Location location) {
                        Log.i(TAG, "Got a fix: " + location);
                    }
                });
    }
```

如果定位请求持续得不到反馈，那就要暂停新监听，并在合适的时候调用removeLocationUpdates(...)方法取消定位请求。不过，既然你使用了setNumUpdates(1)，请求发出后就不用管它了。

（注意，调用requestLocationUpdates (...)方法会出错。不要担心，先不管，稍后会处理。）

最后，编码让搜索按钮发出定位请求。覆盖onOptionsItemSelected(...)方法并在其中调用findImage()方法，如代码清单33-16所示。

代码清单33-16　关联使用搜索按钮（LocatrFragment.java）

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    ...
}
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_locate:
            findImage();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

运行应用并点击搜索按钮。如图33-8所示，刚才忽略的错误来捣乱了：Locatr应用意外中止了。

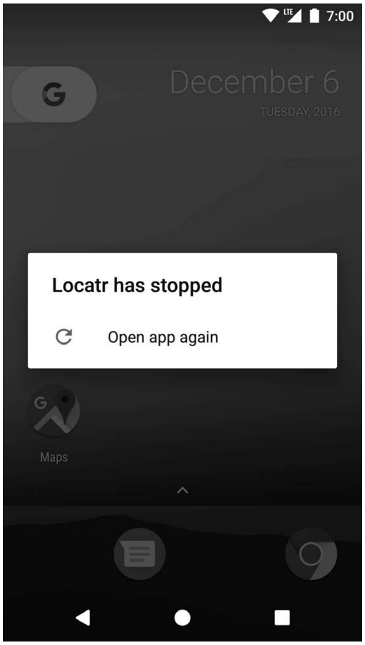


图33-8　使用权限遭拒绝

查看Logcat窗口，可以看到SecurityException抛了出来：

```
FATAL EXCEPTION: main
Process: com.bignerdranch.android.locatr, PID: 7892
java.lang.SecurityException: Client must have ACCESS_FINE_LOCATION permission to
request PRIORITY_HIGH_ACCURACY locations.
    at android.os.Parcel.readException(Parcel.java:1684)
    at android.os.Parcel.readException(Parcel.java:1637)
    ...
    at com.google.android.gms.location.internal.zzd
        .requestLocationUpdates(Unknown Source)
    at com.bignerdranch.android.locatr.LocatrFragment
        .findImage(LocatrFragment.java:102)
```

```
    at com.bignerdranch.android.locatr.LocatrFragment
        .onOptionsItemSelected(LocatrFragment.java:89)
    at android.support.v4.app.Fragment.performOptionsItemSelected(Fragment.java:2212)
    ...
    at java.lang.reflect.Method.invoke(Native Method)
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller
        .run(ZygoteInit.java:886)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:776)
```

看异常可知，`requestLocationUpdates (...)`方法要能工作，你必须获取权限。

33.9　获取运行时权限

处理运行时权限需要做三件事：

- 确认是否拥有权限；
- 获取使用权限，如果还没有的话；
- 监听权限请求反馈。

请求使用权限时，应用会显示标准系统权限获取界面。假如跳出这样的界面，并明确告诉用户，应用为什么要获取某个权限，接下来就是做完上述三件事。

但是，有时候使用权限的理由很难说清。显然，只要搞不清或觉得没必要，用户一般会直接拒绝。这种情况下，应用就需要给个合理性解释。这样的话，这类应用就需要再做一件事：

- 确认是否需给用户个合理的解释。

Locatr应用要使用定位权限的理由不言而喻，所以你不需要额外的解释了。

检查使用权限

首先我们取得权限信息。打开LocatrFragment.java文件，添加一个常量数组，列出应用需要的全部权限，如代码清单33-17所示。

代码清单33-17　添加权限常量（LocatrFragment.java）

```
public class LocatrFragment extends Fragment {
    private static final String TAG = "LocatrFragment";
    private static final String[] LOCATION_PERMISSIONS = new String[] {
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
    };

    private imageView mimageView;
    private GoogleApiClient mClient;
```

在代码中使用的全部Android标准权限都声明在Manifest.permission类里。上述常量数组列出的权限和AndroidManifest.xml里的是一回事。下一步就是请求使用这些权限。

权限安全类型，如dangerous，是赋予权限组而非单个权限的。权限组包含各类具体的使用权限。例如，`ACCESS_FINE_LOCATION`，`ACCESS_COARSE_LOCATION`都属于`LOCATION`权限组。

表33-1 日志级别与方法

权 限 组	权 限
CALENDAR	READ_CALENDAR, WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	READ_CONTACTS, WRITE_CONTACTS, GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE, CALL_PHONE, READ_CALL_LOG, WRITE_CALL_LOG, ADD_VOICEMAIL, USE_SIP, PROCESS_OUTGOING_CALLS
SENSORS	BODY_SENSORS
SMS	SEND_SMS, RECEIVE_SMS, READ_SMS, RECEIVE_WAP_PUSH, RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE

授予权限给权限组内任何单一权限时，同组内的其他所有权限也会获得授权。既然应用需要的ACCESS_FINE_LOCATION和ACCESS_COARSE_LOCATION都属于LOCATION权限组，那只要对其中一种权限操作就可以了。

编写一个私有方法，确认是否能取得LOCATION_PERMISSIONS数组中第一个权限，如代码清单33-18所示。

代码清单33-18 权限检查（LocatrFragment.java）

```
    private void findimage() {
        ...
    }

    private boolean hasLocationPermission() {
        int result = ContextCompat
                .checkSelfPermission(getActivity(), LOCATION_PERMISSIONS[0]);
        return result == PackageManager.PERMISSION_GRANTED;
    }
}
```

checkSelfPermission(...)方法在Marshmallow系统版本中才引入，所以，这里改用了ContextCompat.checkSelfPermission(...)方法。这样既避免了丑陋的版本判断代码，又搞定了兼容问题，一举两得啊。

接下来，在调用findImage(...)方法之前，调用hasLocationPermission()方法确认拥有权限，如代码清单33-19所示。

代码清单33-19 添加权限检查（LocatrFragment.java）

```
@Override
public boolean onOptionsitemSelected(Menuitem item) {
    switch (item.getitemid()) {
        case R.id.action_locate:
```

```
            if (hasLocationPermission()) {
                findimage();
            }
            return true;
        default:
            return super.onOptionsitemSelected(item);
    }
}
```

如果权限检查不通过，就调用`requestPermissions(...)`方法请求授权，如代码清单33-20所示。

代码清单33-20　要求授权（LocatrFragment.java）

```
public class LocatrFragment extends Fragment {
    private static final String TAG = "LocatrFragment";
    private static final String[] LOCATION_PERMISSIONS = new String[ ] {
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
    };
    private static final int REQUEST_LOCATION_PERMISSIONS = 0;
    ...
    @Override
    public boolean onOptionsitemSelected(Menuitem item) {
        switch (item.getitemid()) {
            case R.id.action_locate:
                if (hasLocationPermission()) {
                    findimage();
                } else {
                    requestPermissions(LOCATION_PERMISSIONS,
                            REQUEST_LOCATION_PERMISSIONS);
                }
                return true;
            default:
                return super.onOptionsitemSelected(item);
        }
    }
```

`requestPermissions(...)`是个异步请求方法。调用它之后，Android会弹出系统权限授权对话框要求用户反馈。

为响应用户操作，我们还要写个`onRequestPermissionsResult(...)`响应方法。用户点按ALLOW或DENY按钮后，Android就会调用这个回调方法。再次检查授权结果，如果用户给予授权，就调用`findImage(...)`方法，如代码清单33-21所示。

代码清单33-21　针对授权结果做出反馈（LocatrFragment.java）

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

@Override
```

```
public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                       int[] grantResults) {
    switch (requestCode) {
        case REQUEST_LOCATION_PERMISSIONS:
            if (hasLocationPermission()) {
                findImage();
            }
        default:
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}

private void findimage() {
    LocationRequest request = LocationRequest.create();
    request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
    request.setNumUpdates(1);
```

onRequestPermissionsResult(int, String[ ], int[ ])方法有个参数叫grantResults。如果愿意，也可以查看这个参数值确认授权结果。从上述代码可以看到，我们采用了更方便方式：调用hasLocationPermission()方法时，它里面的checkSelfPermission(...)方法会给出授权结果。所以，只要再调一次hasLocationPermission()方法就可以了。

运行应用并点击搜索按钮。这次，授权对话框跳出来了，如图33-9所示。

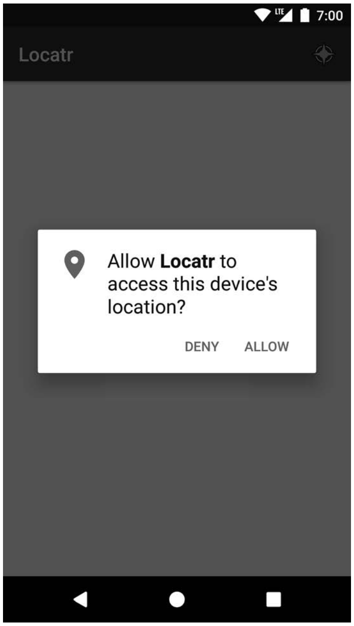


图33-9 授权对话框

如果你按了ALLOW按钮给予授权，除非卸载或关闭权限，应用会一直拥有该授权。如果选了拒绝，应用的授权要求只是暂时被否。下次再点按搜索按钮时，授权对话框还是会跳出来。

33

（为方便调试，建议卸载应用彻底清除授权状态。不要好奇为啥没选关闭授权，相比较而言，卸载应用更简单。）

现在，可以检查定位功能是否符合预期了。如果使用的是模拟器，不要忘了先运行MockWalker应用。（如有应用菜单相关问题，请参阅第13章整合AppCompat库解决。）运行应用并给予授权，然后查看日志，应该看到以下类似数据：

```
...D/libEGL: loaded /system/lib/egl/libGLESv2_MRVL.so
...D/GC: <tid=12423> OES20 ===> GC Version   : GC Ver rls_pxa988_KK44_GC13.24
...D/OpenGLRenderer: Enabling debug mode 0
...I/LocatrFragment: Got a fix: Location[fused 33.758998,-84.331796 acc=38 et=...]
```

在日志中，可以看到定位的数据有经纬度、精确度以及定位时间。在Google地图里输入经纬度数据，就可以看到当前所处的具体位置了（如图33-10所示）。

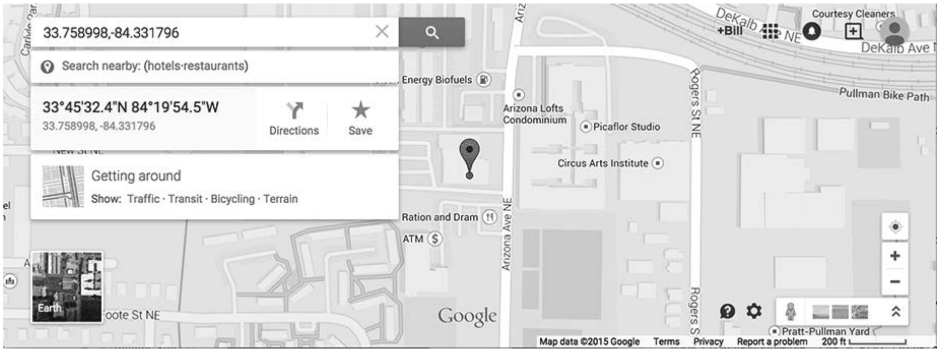


图33-10　当前所处位置

33.10　寻找并显示图片

有了定位数据，下面来使用它。编写一个异步任务找到当前地理位置的`GalleryItem`，下载它关联的图片并显示出来。

编写一个`SearchTask`内部类，执行搜索，如果有返回数据，取出第一个`GalleryItem`。如代码清单33-22所示

代码清单33-22　编写`SearchTask`内部类（LocatrFragment.java）

```
private void findImage() {
    ...
    LocationServices.FusedLocationApi
            .requestLocationUpdates(mClient, request, new LocationListener() {
                @Override
                public void onLocationChanged(Location location) {
                    Log.i(TAG, "Got a fix: " + location);
                    new SearchTask().execute(location);
                }
```

```
            });
    }

    private boolean hasLocationPermission() {
        int result = ContextCompat
                .checkSelfPermission(getActivity(), LOCATION_PERMISSIONS[0]);
        return result == PackageManager.PERMISSION_GRANTED;
    }

    private class SearchTask extends AsyncTask<Location,Void,Void> {
        private GalleryItem mGalleryItem;

        @Override
        protected Void doInBackground(Location... params) {
            FlickrFetchr fetchr = new FlickrFetchr();
            List<GalleryItem> items = fetchr.searchPhotos(params[0]);

            if (items.size() == 0) {
                return null;
            }

            mGalleryItem = items.get(0);

            return null;
        }
    }
```

现在，保存GalleryItem并没有什么用处，但下一章会用到它。

接下来，下载GalleryItem关联的图片数据并解析出图片。然后，在onPostExecute(Void)方法中，使用mImageView显示图片，如代码清单33-23所示。

代码清单33-23 下载并显示图片（LocatrFragment.java）

```
    private class SearchTask extends AsyncTask<Location,Void,Void> {
        private GalleryItem mGalleryItem;
        private Bitmap mBitmap;

        @Override
        protected Void doInBackground(Location... params) {
            ...
            mGalleryItem = items.get(0);

            try {
                byte[] bytes = fetchr.getUrlBytes(mGalleryItem.getUrl());
                mBitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
            } catch (IOException ioe) {
                Log.i(TAG, "Unable to download bitmap", ioe);
            }
            return null;
        }

        @Override
        protected void onPostExecute(Void result) {
```

33

```
            mImageView.setImageBitmap(mBitmap);
        }
    }
```

这样，就应该能在Flickr上找到附近的图片了，如图33-11所示。启动Locatr应用并点击你的位置按钮吧。

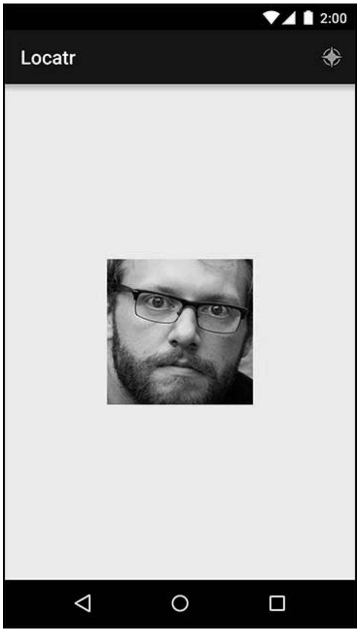


图33-11　应用基本可用了

33.11　挑战练习：权限使用理由

前面说过，很多应用的用户授权对话框给出的权限请求理由让人费解。这种情况下，应用就需要再给用户个合理的解释。

如何处理权限使用请求，Android目前的一套流程是这样的。

(1) 如果用户首次询问授权理由，就跳出系统授权对话框。

(2) 如果用户问了再问，先弹出应用授权解释对话框，然后弹出系统授权对话框。

(3) 如果用户要求永久性拒绝，不显示任何授权对话框。

怎么样，这套流程似乎有点复杂啊。没事，Android已准备好一个便利方法：`ActivityCompat.shouldRequestPermissionRationale(...)`。在第一次权限请求之前，这个方法返回的结果是false；在首次拒绝权限请求后返回true；在用户永久性拒绝权限请求时返回false。

现在，请实施一个权限使用解释对话框（`DialogFragment`），向用户展示消息：“Locatr使

用定位数据寻找你附近发布在Flickr的照片”。调用requestPermissions(...)方法前，使用shouldShowRequestPermissionRationale(...)方法确认是否应该显示权限使用解释对话框。如果已显示，Locatr应用应等用户消除对话框之后再调用requestPermissions(...)方法。（提示：覆盖DialogFragment.onCancel(...)方法可知道用户何时会消除权限使用解释对话框。）

33.12　挑战练习：进度指示器

点击搜索按钮后，Locatr应用接下来干了什么，用户一无所知。一个好的应用应该在用户操作时立即给予醒目反馈。

请优化Locatr应用，在用户点击搜索按钮后，立即显示一个搜索进度条。可使用ProgressDialog类来展示一个旋转的进度指示。另外，还应跟踪SearchTask的运行状态，在有搜索结果返回时，立即清除状态指示。

第 34 章

使用地图

34

在本章中，我们将继续完善`LocatrFragment`。除了搜索并显示附近的图片外，还要找到图片的经纬度数据并在地图上标出。

34.1 导入 Play 地图服务库

继续学习之前，先导入地图库。这是另一个Play服务库。新添加的依赖库名为`com.google.android.gms:play-services-maps:10.0.1`，输入时不要搞错。

34.2 Android 上的地图服务

能获取定位数据，确定手机所处位置已足以让用户开心了；如果数据还能以图形化的方式直观显示，那就更好了。地图应用应该是智能手机上的首个杀手级应用。这也是Android从一开始就有地图服务的原因。

地图服务庞大繁杂，需要大型支持服务器系统来提供基础地图数据。大多数Android服务或应用都能独立为Android开源项目，唯独地图不太可能。

因此，虽然Android一直内置地图服务，但地图API却一直以独立的Android API存在。当前版本的Maps v2 API和Fused Location Provider一起，都包含在Google Play服务里。所以，要使用它，要么有台安装了Play商店应用的设备，要么使用带有Google APIs的模拟器。

如果你在开发地图应用，碰巧翻到本章想看看有什么可以参考的，那么在继续之前，首先要确保已做到了这些（已在上一章完成）：

- 确保设备支持Play服务；
- 已导入合适的Play服务库；
- 确认设备安装上最新版本的Play商店应用（在代码里使用`GoogleAPIAvailability`）。

34.3 获取 Maps API key

使用Maps API还需要在manifest文件中声明你自己的API key。这个API key可以授权你的应用使用Google地图服务。

Google的API key获取流程一直在变，所以，具体如何操作，你需要参阅Google文档developers.google.com/maps/documentation/android/start。

撰写本书时，为获取API key，Google要求首先要创建一个新项目。没搞错吧，我们已经有了Locatr这个完整项目。没事，这难不倒我们，小小变通一下就行了。右键单击com.bignerdranch.android.locatr包，选择NEW→Activity→Gallery...菜单项，然后选择Google Maps Activity创建一个地图activity模板。Activity使用默认名称就好了。

完成之后，Locatr项目的manifest文件会自动添加一些内容，同时还会得到values/google_maps_api.xml这个新文件以及一个新的`MapsActivity`。查看google_maps_api.xml文件，根据其中的指示取得一个可用的API key。最后，在google_maps_api.xml文件中，你的google_maps_key属性值应该和下面的示例差不多，当然key值肯定是不一样的啦。

```
<!-- Our signing key (not yours) -->
<string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">
        AIzaSyClrnnYZEx0iYmJkic0K4rdObrXcFkll-U</string>
```

注意，刚才创建的`MapsActivity`是没用的。所以，如代码清单34-1所示，从项目里以及manifest文件里把它删除。

代码清单34-1　删除`MapsActivity`声明项（Androidmanifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.locatr">
    ...
    <application ...>
        ...
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key"/>

        <activity
            android:name=".MapsActivity"
            android:label="@string/title_activity_maps">
        </activity>
    </application>

</manifest>
```

另外，创建模板xml文件还会自动引入com.google.android.gms:play-services依赖库。我们用不到这个库，而且它还是个巨无霸，会立即让Locatr项目超出方法个数限制，导致项目编译失败。（普通单个应用APK包最多只支持65536个方法。如何突破这个限制，已超出本书讨论范畴，建议参考相关高级主题。）

如代码清单34-2所示，删除不需要的依赖项。

代码清单34-2　删除Play Services依赖项（app/build.gradle）

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
```

```
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.0.1'
    compile   'com.google.android.gms:play-services-location:10.0.1'
    compile   'com.google.android.gms:play-services-maps:10.0.1'
    compile 'com.google.android.gms:play-services:10.0.1'
    testCompile  'junit:junit:4.121
}
```

地图API key搞定了，下面开始学习创建和使用地图。

34.4　创建地图

首先来创建地图。地图显示在MapView中。MapView的使用和其他视图类差不多，但有一点要注意：你必须按如下方式转发所有的生命周期方法。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mMapView.onCreate(savedInstanceState);
}
```

这简直太麻烦了。不过，我们可以找MapFragment帮忙，或者使用支持库版SupportMap-Fragment，痛苦的事就交给SDK去做吧。MapFragment会为我们创建和托管MapView，当然还包括前面说过的生命周期方法回调。

要使用SupportMapFragment，首先要废除原来的用户界面。尽管这听上去像个大工程，实际做起来却很简单。只要改为继承SupportMapFragment类，删除onCreateView(...)方法，再删除所有使用ImageView的代码就行了，如代码清单34-3所示。

代码清单34-3　改用SupportMapFragment（LocatrFragment.java）

```
public class LocatrFragment extends SupportMapFragment Fragment{
    private static final String TAG = "LocatrFragment";
    private static final String[] LOCATION_PERMISSIONS = new String[]{
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
    };
    private static final int REQUEST_LOCATION_PERMISSIONS = 0;

    private ImageView mImageView;
    private GoogleApiClient mClient;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_locatr, container, false);

        mImageView = (ImageView) v.findViewById(R.id.image);

        return v;
```

```
        }
        ...
        private class SearchTask extends AsyncTask<Location,Void,Void> {
            ...
            @Override
            protected void onPostExecute(Void result) {
                mImageView.setImageBitmap(mBitmap);
            }
        }
    }
```

`SupportMapFragment`自己会覆盖`onCreateView(...)`方法，改造任务就这么完成了。运行Locatr应用，可以看到如图34-1所示的地图。



图34-1　一幅极简地图

34.5　获取更多地理位置数据

为在地图上标注图片，需要知道图片的地理位置。再给Flickr API查询串添加一个extra参数，为`GalleryItem`取回经纬度值，如代码清单34-4所示。

代码清单34-4　添加经纬度查询参数（FlickrFetchr.java）

```
    private static final String API_KEY = "yourApiKeyHere";
    private static final String FETCH_RECENTS_METHOD = "flickr.photos.getRecent";
    private static final String SEARCH_METHOD = "flickr.photos.search";
    private static final Uri ENDPOINT = Uri
            .parse("https://api.flickr.com/services/rest/")
```

```
        .buildUpon()
        .appendQueryParameter("api_key", API_KEY)
        .appendQueryParameter("format", "json")
        .appendQueryParameter("nojsoncallback", "1")
        .appendQueryParameter("extras", "url_s,geo")
        .build();
```

现在，为GalleryItem添加经纬度属性，如代码清单34-5所示。

代码清单34-5　添加经纬度属性（GalleryItem.java）

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;
    private double mLat;
    private double mLon;
    ...
    public Uri getPhotoPageUri() {
        return Uri.parse("http://www.flickr.com/photos/")
                .buildUpon()
                .appendPath(mOwner)
                .appendPath(mid)
                .build();
    }

    public double getLat() {
        return mLat;
    }

    public void setLat(double lat) {
        mLat = lat;
    }

    public double getLon() {
        return mLon;
    }

    public void setLon(double lon) {
        mLon = lon;
    }

    @Override
    public String toString() {
        return mCaption;
    }
}
```

然后，从返回的Flickr JSON数据取出经纬度值，如代码清单34-6所示。

代码清单34-6　取出经纬度值（FlickrFetchr.java）

```
private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
        throws IOException, JSONException {
```

```
    JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
    JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

    for (int i = 0; i < photoJsonArray.length(); i++) {
        JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

        GalleryItem item = new GalleryItem();
        item.setId(photoJsonObject.getString("id"));
        item.setCaption(photoJsonObject.getString("title"));

        if (!photoJsonObject.has("url_s")) {
            continue;
        }

        item.setUrl(photoJsonObject.getString("url_s"));
        item.setOwner(photoJsonObject.getString("owner"));
        item.setLat(photoJsonObject.getDouble("latitude"));
        item.setLon(photoJsonObject.getDouble("longitude"));

        items.add(item);
    }
}
```

搞定了图片地理位置信息，接下来在主fragment中添加一些实例变量用来保存搜索结果，一个用于保存待显示的Bitmap，一个用于GalleryItem，还有一个用于Location地理位置，如代码清单34-7所示。

代码清单34-7　添加地图数据（LocatrFragment.java）

```
public class LocatrFragment extends SupportMapFragment {
    ...
    private static final int REQUEST_LOCATION_PERMISSIONS = 0;
    private Bitmap mMapImage;
    private GalleryItem mMapItem;
    private Location mCurrentLocation;
    ...
```

然后，在SearchTask类中，保存这些地图数据，如代码清单34-8所示。

代码清单34-8　保存查询结果（LocatrFragment.java）

```
private class SearchTask extends AsyncTask<Location,Void,Void> {
    private Bitmap mBitmap;
    private GalleryItem mGalleryItem;
    private Location mLocation;

    @Override
    protected Void doInBackground(Location... params) {
        mLocation = params[0];
        FlickrFetchr fetchr = new FlickrFetchr();
        ...
    }

    @Override
```

```
    protected void onPostExecute(Void result) {
        mMapImage = mBitmap;
        mMapItem = mGalleryItem;
        mCurrentLocation = mLocation;
    }
}
```

至此，要用到的地图数据都有了。接下来的任务就是在地图上显示出来。

34.6　使用地图

SupportMapFragment会创建MapView，而MapView又会去托管真正做事的GoogleMap。所以，我们需要引用GoogleMap对象。调用getMapAsync(OnMapReadyCallback)方法可以获取到GoogleMap对象，如代码清单34-9所示。

代码清单34-9　获取GoogleMap（LocatrFragment.java）

```
public class LocatrFragment extends SupportMapFragment {
    ...
    private static final int REQUEST_LOCATION_PERMISSIONS = 0;

    private GoogleApiClient mClient;
    private GoogleMap mMap;
    private Bitmap mMapImage;
    private GalleryItem mMapItem;
    private Location mCurrentLocation;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);

        mClient = new GoogleApiClient.Builder(getActivity())
                ...
                .build();

        getMapAsync(new OnMapReadyCallback() {
            @Override
            public void onMapReady(GoogleMap googleMap) {
                mMap = googleMap;
            }
        });
    }
```

顾名思义，SupportMapFragment.getMapAsync(...)方法可以异步获取地图对象。如果在onCreate(Bundle)方法中调用这个方法，地图一旦完成创建和初始化，你就能取到它。

既然有了GoogleMap对象，根据LocatrFragment的当前状态，就可以更新地图的展示了。首先是放大显示某个目标区域。另外，最好在目标区域周围留出一定间距，所以再添加一个间距尺寸值，如代码清单34-10所示。

代码清单34-10 添加间距（res/values/dimens.xml）

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="map_inset_margin">100dp</dimen>
</resources>
```

然后，添加一个updateUI()方法执行地图放大动作，如代码清单34-11所示。

代码清单34-11 放大（LocatrFragment.java）

```
private boolean hasLocationPermission() {
    ...
}

private void updateUI() {
    if (mMap == null || mMapImage == null) {
        return;
    }

    LatLng itemPoint = new LatLng(mMapItem.getLat(), mMapItem.getLon());
    LatLng myPoint = new LatLng(
            mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude());

    LatLngBounds bounds = new LatLngBounds.Builder()
            .include(itemPoint)
            .include(myPoint)
            .build();

    int margin = getResources().getDimensionPixelSize(R.dimen.map_inset_margin);
    CameraUpdate update = CameraUpdateFactory.newLatLngBounds(bounds, margin);
    mMap.animateCamera(update);
}

private class SearchTask extends AsyncTask<Location,Void,Void> {
```

下面来详细解读上述代码。为四处移动GoogleMap，需要创建了一个CameraUpdate对象。CameraUpdateFactory有好几个静态方法可用。通过改变位置、放大级别以及一些其他属性，可以使用这些静态方法创建不同的CameraUpdate对象。

这里，我们创建了一个指向特定LatLngBounds的CameraUpdate对象。可以把LatLngBounds看作一个包围某个坐标的矩形框。指定西南角和东北角的坐标，就能显式地创建一个LatLngBounds。

通常来讲，提供矩形框包围的一系列坐标会更容易些。LatLngBounds.Builder可以轻松做到这一点：只要创建一个LatLngBounds.Builder，然后针对LatLngBounds要包围的每一个坐标调用.include(LatLng)方法。最后，再调用build()方法，就能得到一个配置好的LatLngBounds。

搞定了包围框，就看如何缩放地图了。有两种方式可选：moveCamera(CameraUpdate)或animateCamera(CameraUpdate)。以动画的方式更有趣些，所以自然是选择animateCamera(CameraUpdate)方法。

接下来，安排在两个地方调用updateUI()方法：首次获得地图时和搜索完成时。如代码清单34-12所示。

代码清单34-12　调用updateUI()方法（LocatrFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    getMapAsync(new OnMapReadyCallback() {
        @Override
        public void onMapReady(GoogleMap googleMap) {
            mMap = googleMap;
            updateUI();
        }
    });
}
...
private class SearchTask extends AsyncTask<Location,Void,Void> {
    ...
    @Override
    protected void onPostExecute(Void result) {
        mMapImage = mBitmap;
        mMapItem = mGalleryItem;
        mCurrentLocation = mLocation;

        updateUI();
    }
}
```

运行Locatr应用并点击搜索按钮。地图应该出现并放大了包含你当前所在位置的一块区域，如图34-2所示。（模拟器用户需要先运行MockWalker应用。）



图34-2　放大版地图

在地图上绘制

地图看上去真不错，但也仅仅是副地图而已。当前，你肯定正处在这幅地图的某个位置，Flickr图片也是。可是，到底在哪呢？好啦，直接在地图上标注出来吧。

在地图上绘制和在普通视图上绘制是两个概念，前者相对更容易。在普通视图上绘制是绘制像素到屏幕上，而地图绘制则是在某个地理位置区域添加对象。使用“绘制”这个字眼，我们的解读是：“首先创建一些对象，然后添加到GoogleMap上，GoogleMap随后会自动绘制出它们。”

其实上面的说法也不够准确。事实上，添加到GoogleMap上的对象也是由GoogleMap创建的。你创建的对象是用来告诉GoogleMap你想要创建什么的。这是种描述性对象，又称options objects。

创建两个MarkerOptions对象。然后，调用mMap.addMarker(MarkerOptions)方法在地图上添加它们，如代码清单34-13所示。

代码清单34-13 地图标注（LocatrFragment.java）

```
private void updateUI() {
    ...
    LatLng itemPoint = new LatLng(mMapItem.getLat(), mMapItem.getLon());
    LatLng myPoint = new LatLng(
            mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude());

    BitmapDescriptor itemBitmap = BitmapDescriptorFactory.fromBitmap(mMapImage);
    MarkerOptions itemMarker = new MarkerOptions()
            .position(itemPoint)
            .icon(itemBitmap);
    MarkerOptions myMarker = new MarkerOptions()
            .position(myPoint);

    mMap.clear();
    mMap.addMarker(itemMarker);
    mMap.addMarker(myMarker);

    LatLngBounds bounds = new LatLngBounds.Builder()
    ...
}
```

调用addMarker(MarkerOptions)方法时，GoogleMap会创建Marker实例并添加到地图上。如果需要删除或修改Marker，应该保存这个Marker实例。每次调updateUI()方法时，都会清除地图，所以这里就不需要保存它了。

运行Locatr应用并点击搜索按钮。可以看到地图上的两个标注出现了，如图34-3所示。

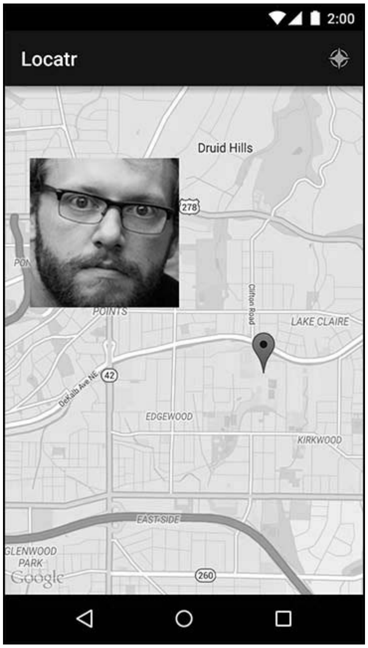


图34-3　带标注的地图

终于，Locatr应用完工了。借助这个应用，你知道该如何使用定位和地图这两个Play服务了，你定位过手机位置，登记使用过Google的Web services API，还学会了地图标注。辛苦了，可以休息一会儿了。

34.7　深入学习：团队开发和 API key

团队合作开发使用API key的应用时，管理API key就会成为一件麻烦事。你的签名凭证会保存在你唯一拥有的密钥文件中。团队成员也同样拥有他们自己的密钥文件。每当有新人加入，都得找他们要SHA1，然后更新你自己的API key凭证。

虽然麻烦，但这起码是个可用的API key管理办法：在项目中管理所有的签名hash值。如果你想借此明确地控制团队成员的开发行为，这未尝不是个合适的办法。

但是，这里还有另外一个方法推荐：创建项目专用调试密钥文件。这个方法首先需要使用Java的`keytool`创建一个新的调试密钥文件，如代码清单34-14所示。

代码清单34-14　创建新的调试密钥文件（终端）

```
$ keytool -genkey -v -keystore debug.keystore -alias androiddebugkey \
--storepass android -keypass android -keyalg RSA -validity 14600
```

命令执行后，`keytool`会抛出一堆问题。据实回答即可。（既然是调试key，除了名字外，其他都可以使用默认值。）

```
$ keytool -genkey -v -keystore debug.keystore -alias androiddebugkey \
--storepass android -keypass android -keyalg RSA -validity 14600
What is your first and last name?
  [Unknown]:  Bill Phillips
...
```

得到了debug.keystore文件后，把它移到应用模块目录。然后，打开项目结构界面，选择app模块，再选择Signing选项页。点击+按钮添加签名配置。Name栏位输入debug，Store File栏位输入debug.keystore，如图34-4所示。

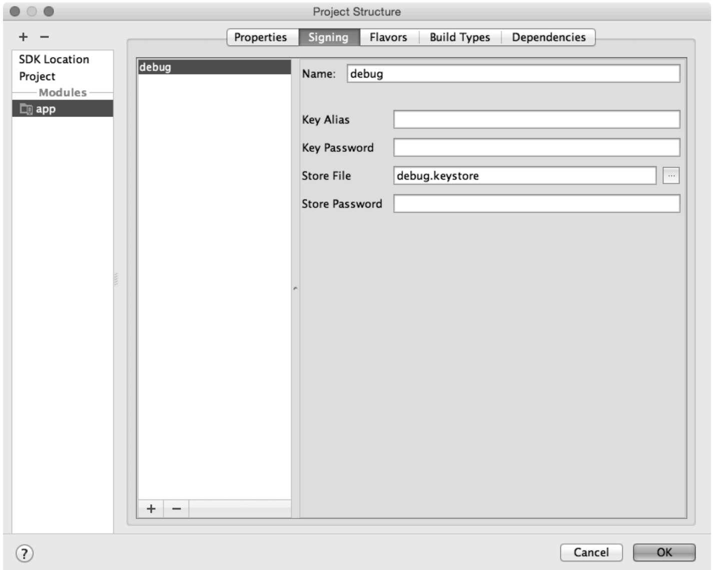


图34-4　配置debug签名key

经过这样的配置，任何人只要使用同样的keystore文件，就能使用同样的API key了。是不是很方便呢？

注意，如果采用这种方式管理API key，在分发debug.keystore时就要当心了。如果仅在团队内部分享，问题不大。但千万不要把keystore发布到公共仓库。否则，任何人都可以使用你的API key了。

第 35 章 material design

Android 5.0（Lollipop）最大的变化是引入了全新的material design设计风格。这种新的视觉设计语言一经推出，立即引爆了设计界。为方便学习和开发，Google还提供了详尽的设计指南。

当然，开发人员一般不怎么关心设计问题。他们认为，设计素材是什么并不重要，只要在应用里用好就行了。然而，随着material design的推出，老观念或许要改一改了。除了全新的用户交互概念，material design还要求开发者具备一定的设计敏感性。一旦熟悉，应用实施定能更加得心应手。

和前面不同，本章可看作本书最大的深入学习专题。所以，本章不涉及实例开发，而且稍后要介绍的大多数内容是可以选读的。

对设计者来说，material design强调以下三大设计原则。

- 实体隐喻（拟物化）：应用部件应具有实物感。
- 醒目、形象、有目的性：如设计精良的杂志或图书带来的体验一样，应用设计元素也应有跃然纸上的观感。
- 动画要有表现力：应用应能动态响应用户操作。

醒目、形象及有目的性这条原则适用于设计人员，本书不作讨论。如果你是个全能型人才，打算自己设计应用，请参阅material design指导手册把握该原则：developer.android.com/design/material/index.html。

就实体隐喻原则来讲，设计时，如果开发人员能参与，设计人员就能创造出更好的拟物化界面。因为开发者知道如何使用Z轴属性布置三维界面，以及如何使用工具栏，浮动操作栏（floating action bar）和snackbar这些新的material组件。

最后是动画要有表现力这条原则。为更好地把握该原则，你需要学习一些新的动画工具：state list animator、animated state list drawable（是的，你没看错，state list animator和animated state list drawable是不同的工具）、circular reveal以及shared element transition。这些工具能给应用带来许多令人艳羡的视觉动画特效。

35.1 material surface

作为开发人员，material design中的material surface是要重点掌握的关键概念。在设计者眼中，material surface如同1dp厚的纸板。这些纸板如同真的纸墨那样，可以变大，也可以显示动画和动态文字，如图35-1所示。



图35-1　有两个material surface的界面

然而，在平面世界里，如同真实纸张，虚拟的material surface再怎么神奇，它的行为表现也无法突破物理限制。例如，一张纸绝不可能穿过另一张纸。同理，创造动画特效时，material surface也要遵守这一原则。

不过，在三维空间里，就可以自由摆放surface以及让它们能够互动。以使用者手指为参照对象，surface可以上下移动，靠近或远离，如图35-2所示。

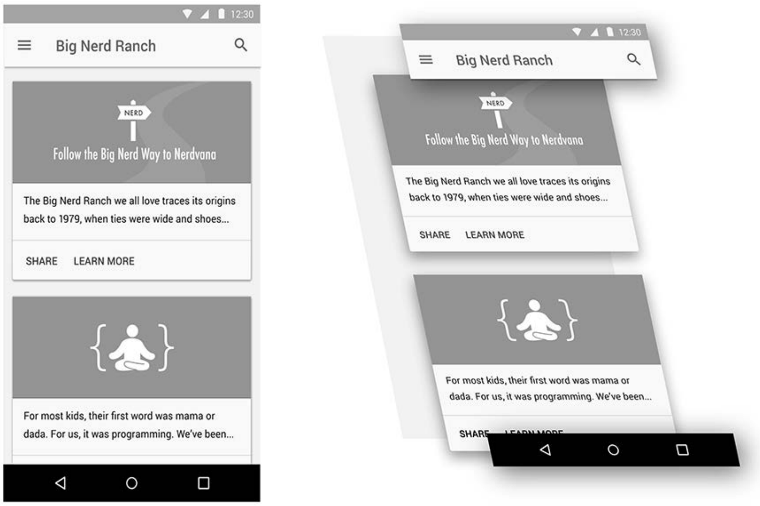


图35-2　三维空间里的material design

要制作一个surface越过另一个的动画特效，可直接向上移动它，然后越过另一个surface，如图35-3所示。

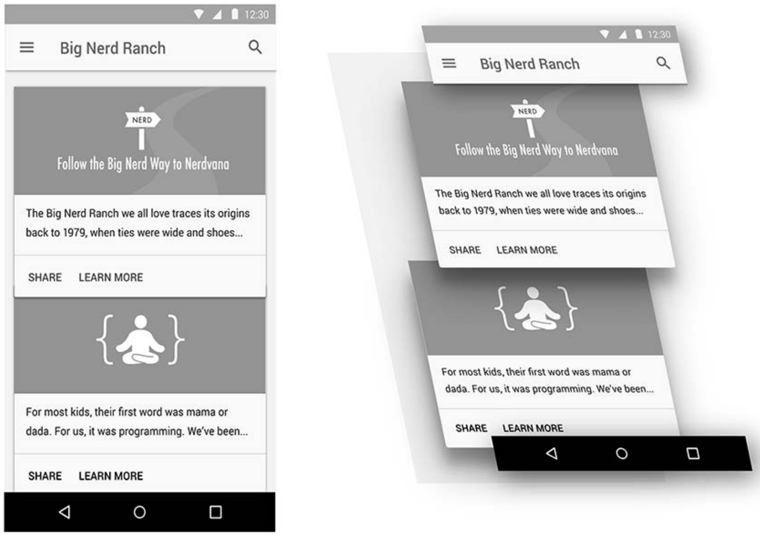


图35-3　surface越过动画

35.1.1　elevation 和 Z 值

应用界面元素间的投影最能让用户准确感知用户界面的深度。投影该如何绘制呢？有些人的第一反应是：这是设计人员的事，开发人员直接用就行了。而且认为这是天经地义的事（不敢苟同）。

稍加分析可知，哪怕是简单的应用，也涉及大量的surface动画特效，让设计者人工绘制这样千变万化的投影几乎不可能。实际上，只要给每个视图设置好elevation，Android就可以帮我们实现阴影绘制。

随着Lollipop系统的发布，Android为布局系统引入了Z轴概念。这允许我们在三维空间里布置视图。如图35-4所示，elevation类似赋予布局视图的坐标：视图可以动态远离其原始坐标，但其原始位置是不变的。

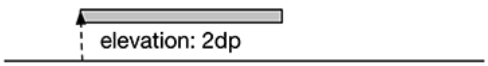


图35-4　Z平面上的elevation

elevation值可以在`View.setElevation(float)`方法或布局XML文件中设置，如代码清单35-1所示。

代码清单35-1 在布局文件中设置elevation值

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:elevation="2dp"/>
```

因为elevation值要作为Z基准值使用，所以最好采用设置XML属性值的方式。而且，相比`setElevation(float)`方法，这种方式更灵活，Lollipop以前版本的系统会默认忽略`android:elevation`属性，因此，难以对付的兼容性问题也就不用考虑了。

要修改View视图的elevation，可以使用`translationZ`和Z属性。它们的用法和第32章介绍的`translationX`、`translationY`、X和Y一样。

如图35-5所示，Z值总是等于elevation加上`translationZ`。如果给Z一个值，那么系统会自动算出`translationZ`值。

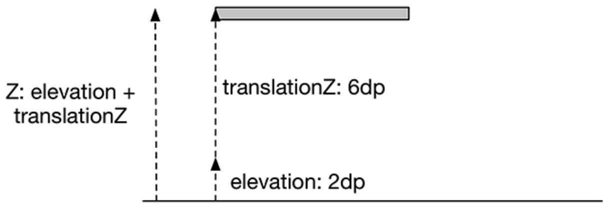


图35-5 Z和translationZ

35.1.2 state list animator

使用material design的应用通常包含许多用户交互动画。例如，在Lollipop设备上，按住按钮时，按钮会基于Z轴向用户手指靠近，松开时则会后退。

为方便实现这样的动画特效，Lollipop引入了state list animator。它是state list drawable的动态版竞争对手：state list drawable只是在切换素材资源（例如，点按按钮，按钮变灰），而state list animator能赋予视图动画特效。只要在res/animator中定义一个state list animator，就能实现按钮浮出的动画效果，如代码清单35-2所示。

代码清单35-2 state list animator 使用示例

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true">
        <objectAnimator android:propertyName="translationZ"
                        android:duration="100"
                        android:valueTo="6dp"
                        android:valueType="floatType"
                        />
    </item>
    <item android:state_pressed="false">
        <objectAnimator android:propertyName="translationZ"
                        android:duration="100"
```

```
                        android:valueTo="0dp"
                        android:valueType="floatType"
                />
        </item>
    </selector>
```

对属性动画来说，这非常有用。假设想实施帧动画，可以使用另一个动画工具：animated state list drawable。

这个工具的名字很容易迷惑人。听起来和state list animator差不多，但实际用法完全不一样。类似于常规的state list drawable，使用animated state list drawable可以为视图的不同状态定义不同的图片，甚至能定义状态间的帧动画转场。

还记得吗？第23章的BeatBox应用的声音按钮是使用state list drawable定义的。如果你是个设计狂，没准还想实现每次按钮按下时播放多帧动画。没问题，按代码清单35-3修改XML文件就可以了。不过，修改版XML文件必须放在res/drawable-21目录中，因为Lollipop之前的系统不支持这个功能。

代码清单35-3　animated state list drawable使用示例

```
<animated-selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/pressed"
          android:drawable="@drawable/button_beat_box_pressed"
          android:state_pressed="true"/>
    <item android:id="@+id/released"
          android:drawable="@drawable/button_beat_box_normal" />

    <transition
        android:fromId="@id/released"
        android:toId="@id/pressed">
        <animation-list>
            <item android:duration="10" android:drawable="@drawable/button_frame_1" />
            <item android:duration="10" android:drawable="@drawable/button_frame_2" />
            <item android:duration="10" android:drawable="@drawable/button_frame_3" />
            ...
        </animation-list>
    </transition>
</animated-selector>
```

注意，在上述代码中，`animated-selector`元素里的每个item都有ID。在不同的ID间定义转场就可以播放多帧动画。如果还想实现按钮释放动画，那就再添加一个`transition`标签。

35.2　动画工具

material design引入了很多漂亮的动画特效。有些很容易实现，有些需要花点力气。不过，也不用担心，Android为你准备了便利工具。

35.2.1 circular reveal

circular reveal动画看起来就像墨滴在一张纸上快速扩散。从一个交互点出发（通常是用户的按压点），视图或是一段文字向外扩散显现。具体效果如图35-6所示。



图35-6 在BeatBox应用中模拟circular reveal动画特效

还记得吗？在第6章，为隐藏按钮，我们实现过这种动画特效。这里，我们来看另一个实现版本。与之前相比，这个版本的circular reveal动画特效实现起来繁琐一些。

要创建circular reveal动画特效，可调用ViewAnimationUtils的createCircularReveal(...)方法。该方法有好几个参数：

```
static Animator createCircularReveal(View view, int centerX, int centerY,
                                     float startRadius, float endRadius)
```

第一个View参数就是要向外扩散显现的视图。在图35-6中，这个视图就是和BeatBox-Fragment宽高一致的红色实心视图。如果动画从startRadius（值为0）圆点开始到endRadius结束，这个红点视图会先变为透明状态，然后随着一个不断放大的圆慢慢显现。centerX和centerY是这个圆的圆点坐标（也就是View的坐标）。该方法会返回一个Animator（和第32章用过的Animator一样）。

material design指南指出，circular reveal动画应该开始于用户手指在屏幕上的触点。所以，首先要找到用户点击视图的坐标，如代码清单35-4所示。

35

代码清单35-4 找到视图点击坐标

```
@Override
public void onClick(View clickSource) {
```

```
    int[] clickCoords = new int[2];

    // Find the location of clickSource on the screen
    clickSource.getLocationOnScreen(clickCoords);

    // Tweak that location so that it points at the center of the view,
    // not the corner
    clickCoords[0] += clickSource.getWidth() / 2;
    clickCoords[1] += clickSource.getHeight() / 2;

    performRevealAnimation(mViewToReveal, clickCoords[0], clickCoords[1]);
}
```

然后开始执行circular reveal动画，如代码清单35-5所示。

代码清单35-5　创建并执行circular reveal动画

```
private void performRevealAnimation(View view, int screenCenterX, int screenCenterY) {
    // Find the center relative to the view that will be animated
    int[] animatingViewCoords = new int[2];
    view.getLocationOnScreen(animatingViewCoords);
    int centerX = screenCenterX - animatingViewCoords[0];
    int centerY = screenCenterY - animatingViewCoords[1];

    // Find the maximum radius
    Point size = new Point();
    getActivity().getWindowManager().getDefaultDisplay().getSize(size);
    int maxRadius = size.y;

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        ViewAnimationUtils.createCircularReveal(view, centerX, centerY, 0, maxRadius)
                .start();
    }
}
```

注意，成功调用`createCircularReveal(...)`方法有个前提条件：布局中已有目标视图。

35.2.2　shared element transition

shared element transition（又称为hero transition）是material design中引入的另一新动画特效。它适用于一种特殊场景：两个视图显示相同元素。

前面，在CriminalIntent应用中，crime记录的明细界面会显示一张缩略图。当时，有个挑战练习要求弹出一个新视图展示全尺寸照片。假如做过的话，你应还有印象，完成后的效果图差不多如图35-7所示。

这是一种常见的交互模式：点击一个元素，弹出新的视图显示元素明细。

对于两个展示相同元素的视图，其间的任何切换动画都可以用shared element transition实现。图35-7中，右边的大图片和左边的小图片是同一张图。所谓相同元素就是指这张图。它实际就是个共享元素（shared element）。

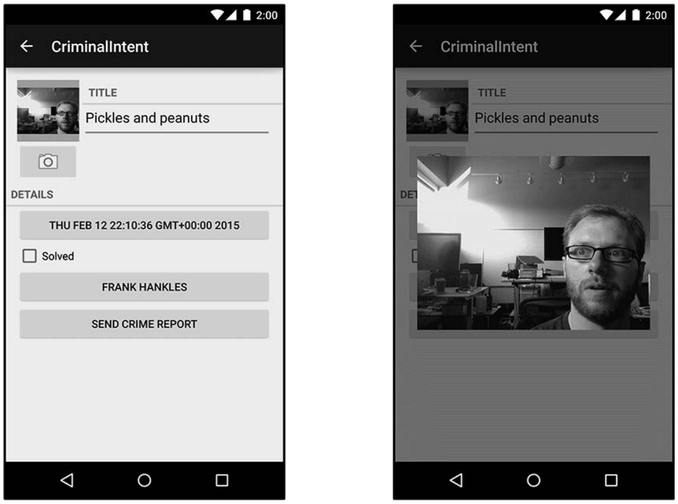


图35-7 放大版照片视图

自Lollipop开始，Android终于有办法实现activity或fragment间的转场动画了。图35-8是动画过程中的一幅截图，可大致看出实现效果。下面学习如何把它应用于activity。

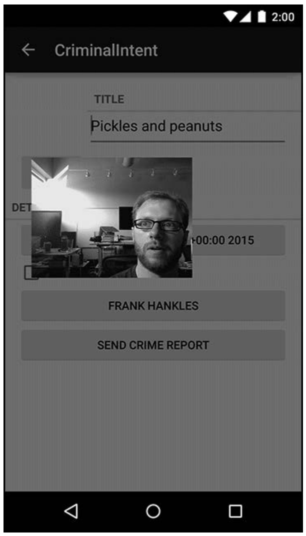


图35-8 shared element的变换

实现activity间的转场动画涉及以下三个步骤。

(1) 打开activity transition。

(2) 为每个shared element视图设置transition名称。

(3) 启动带`ActivityOptions`（触发动画）的另一个activity。

首先是打开activity transition。如果你的activity使用了AppCompat主题，这个步骤可以直接跳过。（AppCompat继承Material主题， activity transition已打开。）

在CriminalIntent的例子中，为了让目标activity拥有透明背景，我们使用了@android:style/Theme.Translucent.NoTitleBar主题样式。这个主题没有继承Material主题，所以需要手工打开activity transition。有两种方式可以打开activity transition，先来看如何用代码打开它，如代码清单35-6所示。

代码清单35-6　以代码的方式打开activity transition

```
@Override
public void onCreate(Bundle savedInstanceState) {
    getWindow().requestFeature(Window.FEATURE_ACTIVITY_TRANSITIONS);
    super.onCreate(savedInstanceState);
    ...
}
```

另外一种方式是修改activity使用的样式，设置android:windowActivityTransitions属性值为true，如代码清单35-7所示。

代码清单35-7　在样式里打开activity transition

```
<resources>
    <style name="TransparentTheme"
        parent="@android:style/Theme.Translucent.NoTitleBar">
        <item name="android:windowActivityTransitions">true</item>
    </style>

</resources>
```

再来看如何为shared element视图设置transition名称。Android在API 21中为View引入了transitionName属性。所以，可以在布局或代码中设置这个属性值。具体用哪种方式，要看当时的场景。本例中，我们在布局XML文件里，将android:transitionName属性设置为image，如图35-9所示。

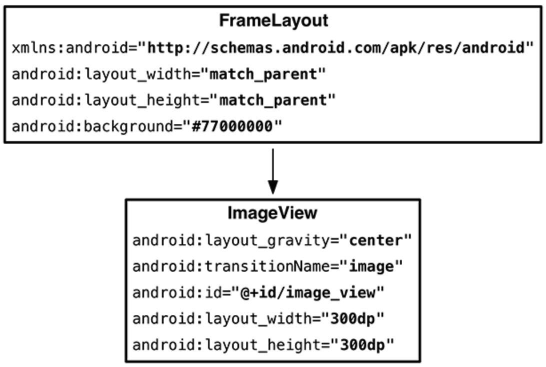


图35-9　设置android:transitionName属性值为image

然后，再定义一个startWithTransition(...)静态方法，为另一视图设置相同的transition名，如代码清单35-8所示。

代码清单35-8　定义startWithTransition(...)方法

```
public static void startWithTransition(Activity activity, Intent intent,
  View sourceView) {
    ViewCompat.setTransitionName(sourceView, "image");
    ActivityOptionsCompat options = ActivityOptionsCompat
        .makeSceneTransitionAnimation(activity, sourceView, "image");

    activity.startActivity(intent, options.toBundle());
}
```

Android旧版本系统中，View视图没有setTransitionName(String)属性方法。所以，需要使用ViewCompat.setTransitionName(View, String)方法。

在代码清单35-8中，作为三个步骤的最后一步，我们创建了ActivityOptions对象，让它告诉操作系统，共享元素（shared element）是哪个，使用什么transitionName值。

transition和shared element transition还有其他用途。例如，它们还可以用于fragment间的转场动画。欲了解更多，请阅读Google的transition框架文档：https://developer.android.com/training/transitions/overview.html。

35.3　新的视图组件

material design指南还设计了一些全新视图组件，它们在Lollipop系统中尚属首秀。Android设计团队已实现了一些。下面就一起来学习一下，没准哪天就要用到。

35.3.1　card

首先学习的是card这个新组件，如图35-10所示。它是个frame容器组件。



图35-10　card组件

card容器组件用来装载其他内容。从图35-10可以看出，它的边角有点圆，微微浮出底平面，在身后投下了一圈阴影。

设计问题已超出本书讨论范畴，所以，什么时候、什么地方该用card，我们给不了建议。（想知道的话，可以查阅Google的material design文档：http://www.google.com/design/spec。）我们只能教你创建它：使用CardView。

类似RecyclerView，CardView是com.android.support:cardview-v7支持库中的一个视图类。使用前，首先要在项目中添加这个依赖。

引入以后，就可以像使用布局中的ViewGroup一样使用CardView了，如代码清单35-9所示。CardView是个FrameLayout子类，所以FrameLayout的任何布局参数它都可以用。

代码清单35-9　在布局中使用CardView

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:tools="http://schemas.android.com/tools"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              tools:context=".MainActivity">
    <android.support.v7.widget.CardView
        android:id="@+id/item"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_margin="16dp"
        >
        ...
    </android.support.v7.widget.CardView>

</LinearLayout>
```

既然是支持库类，自然能较好地兼容旧设备。和其他组件不一样，CardView总带投影效果。（不过，在旧设备上，它只绘制自己，投影效果不明显。）感兴趣的话，可以查阅CardView文档，看看在新旧设备上还有哪些视觉差异。

35.3.2　floating action button

另一个常见组件是floating action button（FAB），如图35-11所示。

floating action button是在Google设计支持库中实现的。同样，使用前，首先要在项目中引入com.android.support:design:24.2.1这个依赖。

floating action button可以看作是由一个实心圆和一个OutlineProvider提供的圆形投影组成。作为ImageView子类，FloatingActionButton类负责生成实心圆和阴影。创建floating action button很简单：在布局文件中放入FloatingActionButton，并设置它的src属性指向要显示在按钮上的图片。



图35-11 一个floating action button

如果把floating action button放在Framelayout中，设计支持库还会引入一个比较智能的CoordinatorLayout。这个FrameLayout子类布局会随其他控件的移动调整floating action button的位置。这样，如果要展示一个Snackbar（稍后介绍），FAB会自动上移以避免被Snackbar挡住。具体实现如代码清单35-10所示。

代码清单35-10 布局floating action button

```
<android.support.design.widget.CoordinatorLayout
             xmlns:android="http://schemas.android.com/apk/res/android"
             xmlns:tools="http://schemas.android.com/tools"
             xmlns:app="http://schemas.android.com/apk/res-auto"
             android:layout_width="match_parent"
             android:layout_height="match_parent">
    [... main content here ...]
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/floating_action_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:layout_margin="16dp"
        android:src="@drawable/play"/>
</android.support.design.widget.CoordinatorLayout>
```

上面这段代码会把按钮放在右下角位置的其他内容之上，互不干扰。

35.3.3 snackbar

snackbar比floating action button复杂一些。它位于屏幕底部，是一种不怎么需要用户交互的控件，如图35-12所示。



图35-12　一个snackbar控件

snackbar控件的动画效果是从屏幕底部向上弹出。短暂停留一会，或是屏幕上有了其他交互操作，它就会自动退出。snackbar的作用和Toast类似。不同的是，snackbar本身就是应用界面的一部分；而Toast会出现在应用界面之上，一动不动，即使用户已导航至其他页面。此外，为方便用户快速操作，snackbar上还允许放置按钮。

和floating action button一样，snackbar也是在Android设计支持库中实现的。

创建和显示snackbar的方式和Toast类似，如代码清单35-11所示。

代码清单35-11　创建并显示snackbar

```
Snackbar.make(container, R.string.munch, Snackbar.LENGTH_SHORT).show();
```

创建Snackbar需要传入放置它的视图、要显示的文字以及它暂留的时间。最后，调用show()方法展示它。

针对某些破坏性操作，可以在snackbar控件右端配置应急操作按钮，比如undo。万一用户不小心犯了错，比如删除了一条crime记录，就可以立即借助它执行反删除。

35.4　深入学习 material design

本章介绍了一大堆有趣实用的新工具。不过，工具不是摆设，不要放那里吃灰，有机会就要用起来。所以，开发人员平时要多多留心，积极研究如何使用这些全新工具和动画特效。

Android 提供有 material design 规格说明书（www.google.com/design/spec/material-design/introduction.html），它里面满是精彩的点子，可以帮助激发设计灵感。当然，也可以从Google Play下载精品应用，看看别人是怎么应用material design的，再问问自己：怎么用在我自己的应用里呢？相信自己，一番努力后，你肯定能做出超越精品的好应用。

第 36 章

编　后　语

36

恭喜你完成本书的学习！这很了不起，不是人人都能做到的。现在就去犒赏一下自己吧！

总之，千辛万苦终有回报：你已经是一名合格的Android开发者了。

36.1　终极挑战

最后，请再接受一个挑战：成为一名优秀的Android开发人员。成为优秀的开发者，可以说是千人千途。每个人都应去找寻最适合自己的路。

那么，路在何方？对此，我们有一些建议。

- 编写代码。现在就开始。如不加以实践，很快就会忘记所学知识。参与开发一些项目，或者自己写个简单应用。无论怎样，不要浪费时间，利用一切机会编写代码。
- 持续学习。读完本书，你已经掌握了Android开发领域的很多知识。你有灵感了吗？利用所学知识，可尝试开发一些自己感兴趣的应用。开发时，如遇到问题，记得经常查阅文档，或阅读更高级主题的图书。另外，也可收看YouTube的Android开发者频道，或收听Google的Android开发者播客（Android Developers Backstage podcast）。
- 参与技术交流。参与本地技术交流大会，多认识些乐于助人的开发者。参与Android开发者大会，与其他Android开发人员（包括我们）面对面交流。另外，还可以关注一些活跃在Twitter和Google Plus上的开发高手。
- 探索开源社区。登录www.github.com，上面有海量的Android开发资源。找找那些很酷的共享库，顺便看看共享者贡献的其他项目资源。同时，也请积极共享自己的代码，如果能帮到别人，那最好不过了。另外，也可以订阅Android每周邮件列表，及时跟踪了解Android开发社区新动向（http://androidweekly.net/）。

36.2　关于我们

来Twitter找我们吧！Bill、Chris和Kristin的帐号分别是@billjings、@cstew和@kristinmars。

有兴趣的话，也可以访问http://www.bignerdranch.com/books，看看我们的其他指导书。同时，我们还为开发者提供课时一周的各类培训课程，可保证在一周内轻松学完。当然，如果有高质量

代码开发需求，我们也可以提供合同开发。欲详细了解，请访问网站http://www.bignerdranch. com。

36.3　致谢

没有读者，我们的工作将毫无意义。感谢所有购买并阅读本书的读者！

权威 —— 源自大名鼎鼎的Big Nerd Ranch训练营培训讲义，该训练营已经为微软、Google、Facebook等行业巨头培养了众多专业人才。

全面 —— 涵盖Android开发所有必备理论概念和技术知识点，从Android 4.4到Android 7.0都适用。

实用 —— 8个Android应用开发实战项目，传授一线开发经验。

易懂 —— 以循序渐进的方式精心编排章节，一步一步写出Android应用。

时新 —— 在前两版的基础上，新增数据绑定、MVVM架构、Android辅助功能等内容。

“在阅读本书之前，我跟着一些Android开发教程自学，却发现困难重重。本书用简单易懂的示例帮助我轻松上手，并让我对Android开发有了更好的整体认识。”

—— Amazon读者

“Android技术更新得太快了。每次拿起讲Android的书，我都担心自己在学快过时的内容。但这本书不一样，我想学的都在书中。此外，我还想给书中的流程图点赞，它们使晦涩的概念变得十分清晰。”

—— Amazon读者

“对于我们来说，这是一本非常全面的培训教材。它已使我们公司数百名工程师掌握了构建Android应用的诀窍。另外，对想要提升Android开发技能的人，这本书同样也有很大帮助。”

—— Mike Shaver，Facebook通信工程主管

“不管你是刚刚迈进Android开发的大门，还是准备掌握更多高级开发技术，本书都非常值得看。其完整的内容体系、清晰的组织结构以及轻松的讲述风格，都让人过目不忘。”

—— James Steele，《Android开发秘籍》作者

Big Nerd Ranch

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机 / 移动开发 / Android

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-45759-2

ISBN 978-7-115-45759-2

定价：129.00元